

Część 1

Wprowadzenie do Java. Teksty w trybie konsolowym.

Podstawowe definicje programowania obiektowego.

Historia oraz genealogia Java.

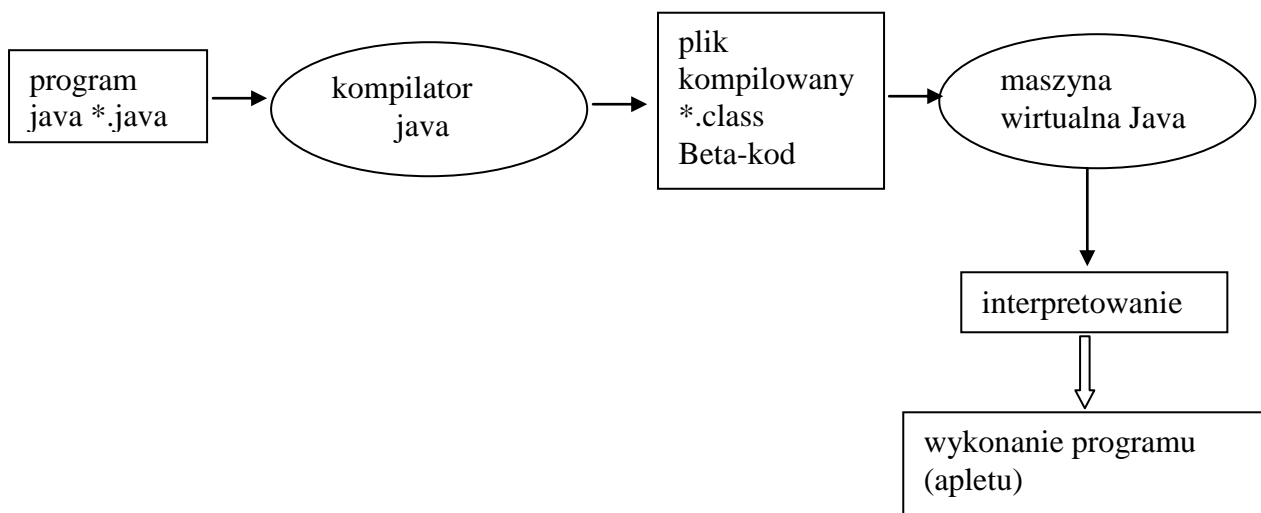
Jedną z największych zalet Javy, jest przenośność; czyli ten sam program, przynajmniej teoretycznie, będzie, bowiem działał na różnych urządzeniach i platformach sprzętowo-systemowych. Będzie go, więc można uruchomić i na komputerze PC, Macintosh, pod systemem Windows i pod Uniksem.

Takie były właśnie początki rozwoju Javy w roku 1990. Język ten został opracowany przez zespół kierowany przez Jamesa Goslinga w firmie Sun Microsystems, choć w rzeczywistości prace były prowadzone na Uniwersytecie Stanford. Projekt gotowy był bardzo szybko, bo już roku 1991, jednakże do 1994 roku zupełnie nie udało się go spopularyzować. Z tego też powodu prace zostały zawieszone.

Historia informatyki rządzi się jednak przypadkiem, były to, bowiem lata ogromniej ekspansji i wręcz eksplozji Internetu. W ten oto sposób w roku 1995 Java ujrzała światło dzienne, a to, co nastąpiło później zaskoczyło chyba wszystkich, łącznie z twórcami języka. Java niezwykle szybko, nawet jak na technologię informatyczną, została zaakceptowana przez społeczność internautów i programistów na całym świecie. Jest to doskonale skonstruowany język programowania, który programistom przypada do gustu zwykle przy pierwszym zetknięciu. Od Javy po prostu nie ma już odwrotu.

Wykonywanie Beta-kod. Maszyna wirtualna Javy, uruchamianie programu.

Programy (Aplety) Javy można wykonać na każdym komputerze, czyli nie powinno być różnicy jaki procesor ma dany komputer czy jaki jest zainstalowany system operacyjny. W tym celu na każdym komputerze na, którym ma wykonany program(aplet) instalowa jest **maszyna wirtualna Javy**, której zadaniem jest interpretowanie plików powstałych po kompilacji, są to pliki z rozszerzeniem *.class nazywane Beta-kod.



Instalacja Java

1) Do instalacji służy plik jdk-6u12-windows-i586-p.exe. Pobierz od nauczyciela i zainstaluj. Nie wykonuj rejestracji.

2) Sprawdź czy instalacja wykonana została prawidłowo poprzez: przejdź do okienka DOSu i wpisz instrukcję java. jeśli system wykona tę instrukcję, początek wyglądu ekranu:

Usage: java [-options].....

3) Dokonaj konfiguracji Javy poprzez:

Panel sterowania → System → Właściwości systemu → zakładka Zaawansowane → przycisk Zmienne środowiskowe → zaznacz pole Zmienna o nazwie Path następnie przycisk Edytuj → przechodzimy na koniec wpisujemy średnik po nim ścieżkę gdzie znajduje się Java np. c:\Program Files\Java\jdk1.6.0_12\bin

4) Przejdź do okienka DOS i wpisz komendę javac. Jeśli zostanie wykonana to poprzedni podpunkt wykonałeś poprawnie.

Narzędzia JDK

Podstawowe narzędzia Java Development Kit:

javac- kompilator Java. Tworzy kod pośredni (bytecode)

java- interpreter Java - wykonuje skompilowane programy w Javie (zapisane już w bytecode)

appletviewer- jest bardzo przydatnym narzędziem pozwalającym wykonywać i debugować applety Java bez użycia przeglądarki.

javadoc- tworzy dokumentację API (w HTML), pobierając komentarze z kodu źródłowego

jdb- narzędzie debugujące Javy

javap- dekompile Java

jar - tworzy pliki JAR (Java Archive). Kompresuje klasy i tworzy jeden plik jar.

Aplet

Programy w Javie mogą być:

- apletami (ang. applet) lub
- aplikacjami (ang. application).

Różnica jest taka, że aplikacja jest programem samodzielnym uruchamianym z poziomu systemu operacyjnego (a dokładniej: maszyny wirtualnej Javy operującej na poziomie

systemu operacyjnego) i tym samym ma pełny dostęp do zasobów udostępnianych przez system.

Aplet jest uruchamiany pod kontrolą innego programu, najczęściej przeglądarki internetowej, i ma dostęp jedynie do środowiska, które mu ten program udostępni. Inaczej aplet to program zagnieżdżony w innej aplikacji

Pisanie programu

- Program pisany jest w dowolnym tekstowym edytorze,
- Nazwa programu nagrywanego na dysku powinna być identyczna jak nazwa programu po słowie public class i mieć rozszerzenie *.java

uwagi odnośnie nazwy programu:

- nazwa programu pisana jest z dużej litery np. Zadanie1000,
- nazwą programu nie może być słowo kluczowe Javy np. class, public,
- nazwa programu nie może rozpoczynać się od cyfry,
- nazwa programu nie może mieć spacji ani polskich znaków.

Tworzenie programu wynikowego

Uwaga

Zarówno przy pisaniu kodu Źródłowego, jak i przy wydawaniu komend w wierszu poleceń, należy zwrócić szczególną uwagę na wielkość wpisywanych liter. Jakakolwiek pomyłka, w większości przypadków, spowoduje, że kompilacja się nie uda.

1)kompilacja programu

javac nazwa_programu.java

uwagi:

- uruchom okienko DOS
- przejdź do folderu gdzie nagrałeś plik źródłowy *.java
- konieczne podanie pliku do kompilacji z rozszerzeniem,
- powstanie plik o rozszerzeniu *.class
- powstały plik *.class może być uruchamiany na różnych platformach

2)uruchomienie

java nazwa_klasy_z_main

uwagi:

- podajemy nazwę klasy, która zawiera metodę main() czyli bez rozszerzenia

3)Uwagi praktyczne o tworzeniu i kompilacji programów w Java:

- używaj Totalcommandera,
- utwórz dla każdego zadania oraz przykładu osobny folder F7,
- edytuj programy poprzez F4,
- przejdź do folderu gdzie zapisałeś program *.java i w dolnym okienku wpisz cmd → a uruchomi się okienko DOS i teraz możesz wydać komendy javac oraz java
- możesz stworzyć plik wsadowy (*.bat). Wpisz treść pliku wsadowego (jaka patrz poniżej) nagraj ten plik wsadowy o nazwie np.w.bat do folderu gdzie znajduje się plik np. Przykład16.java (treść programu, którą chcesz wykonać). Wejdź do folderu gdzie jest plik Przykład16.java i uruchom plik wsadowy poprzez napisanie nazwy pliku wsadowego tutaj „w” i ENTER. Będziesz musiał wcisnąć dwa razy Enter.

Treść pliku w.bat

@echo off

chcp 1250
javac Przyklad16.java
pause
java Przyklad16
pause

Zadanie → praca domowa.

Zapisz w zeszycie treści pytań pod nimi odpowiedzi. Pytanie linijką podkreś na zielono.

Pytanie 1

- Zapisz w zeszycie uwagi o strukturze podstawowej programu w Java (są pod przykładem).

Pytanie 2

zapisz w zeszycie w jaki sposób wyświetlany jest tekst z i bez przechodzenia do nowej linii (jest pod przykładem) oraz co oznacza \n.

Pytanie 3

Pisanie programu (sześć uwag)

Pytanie 4

Polskie znaki na konsoli (sposób pierwszy).

Pytanie 5

Zapisz różne sposoby prezentacji algorytmów(siedem sposobów)

Pytanie 6

Jak należy rozumieć przenośność Javy.

Pytanie 7

Kto i kiedy zapoczątkował prace na Javą.

Pytanie 8

Jakie wydarzenie przyczyniło się do popularyzacji Javy.

Pytanie 9

Co to jest applet?

Pytanie 10

5)Narysuj oraz opisz schemat powstawania Beta-kodu oraz maszyny wirtualnej.

Pytanie 11

Zapisz podstawowe narzędzia Java Development Kit.

Pytanie 12

Co to jest programowanie obiektowe uwzględnij pojęcia:

- obiekt,
- stan,
- pola,
- zachowanie,
- obiektowy program komputerowy.

Pytanie 13

Czym różni się programowanie obiektowe od strukturalnego.

Pytanie 14

Wymień w podpunktach zalety programowania obiektowego w Javie.

Pytanie 15

Czym jest klasa?

Pytanie 16

Podaj teoretyczną deklarację klasy w Java.

a)prosty sposób

b)całościowy sposób deklaracji klasy

Pytanie 17

Co definiujemy wewnątrz klasy.

Pytanie 18

Jaką nazwę uzyskują klasy podczas kompilacji. Jakie uzyskuje rozszerzenie?

Pytanie 19

Co dzieje się po procesie kompilacji, gdy w jednym pliku tekstowym zapisane jest wiele klas.

Pytanie 20

Co to jest modyfikator dostępu do klasy.

Pytanie 21

Znaczenie modyfikatora dostępu do klasy, krótko opisz:

- public
- private
- protected
- private protected
- package

Pytanie 22

Co to jest specyfikator? Krótko opisz:

- abstract
- final
- static

Pytanie 23

Co to jest metody klasy? Co posiada metoda. Podaj składnię definiowania metody po umieszczeniu w ciele klasy.

Pytanie 24

Co oznacza słowo void oraz co należy zrobić gdy metoda nie zwraca żadnej wartości. Podaj w jaki sposób wywołujemy metodę.

Pytanie 25

Do czego służy pole klasy. W jaki sposób są definiowane pola.

Pytanie 26

Co to jest obiekt, w jaki sposób tworzymy obiekt(opisz dokładnie). Podaj praktyczny przykład.

Pytanie 27

Co to jest konstruktor. Kiedy wywoływany jest konstruktor. Jaką nazwę ma konstruktor? Gdy jest więcej konstruktorów, jakie mają nazwy. Jakie są zadania konstruktora? Jaką wartość zwraca konstruktor? Zapisz składnię definiowania konstruktora wewnątrz klasy.

Pytanie 28

Co to jest destruktory. W jaki sposób możemy go użyć w Java.

Pytanie 29

Co to jest hermetyzacja danych?

Pytanie 30

Co to jest dziedziczenie? Zanotuj wszystkie uwagi o dziedziczeniu.

Pytanie 31

Co to jest Poliformizm?

Pytanie 32

Na domowym komputerze zainstaluj Java (możesz na maszynie wirtualnej).

Wykonaj trzy zrzuty ekranu z twoim nazwiskiem (okienka CMD).

- a)działanie instrukcji Java
- b)działanie instrukcji Javac
- c)działanie Przykładu1

Wstaw zrzuty do pliku PDF. Przynieś je na pendrive oraz wyślij dla pewności na pocztę.

KONIEC PYTAŃ PRACY DOMOWEJ.

Różne sposoby prezentacji algorytmów

- opis słowny,
- schemat blokowy,
- lista, kroków,
- drzewo algorytmu,
- drzewo wyrażeń,
- program w języku,
- psedokod.

Przykład1

- Dokonaj kompilacji oraz uruchomienia przykładu.

```
public class Przyklad1
{
    public static void main(String args[])
    {
        System.out.println("jestem w IV klasie");
        System.out.println("matura w maju");
    }
}
```

Wyświetlanie tekstu

System.out.print("tekst");

wyświetlenie tekstu bez przechodzenia do nowej linii

System.out.println("tekst");

wyświetlenie tekstu z przejściem do nowej linii po wyświetleniu tekstu

System.out.println("\ntekst");

wyświetlenie tekstu z przejściem do nowej linii przed i po wyświetleniu tekstu(zwróć uwagę na \n .

Znaki " muszą być z notatnika a nie z Worda

Uwagi o strukturze podstawowej programu w Java.

public→oznacza, że definiowana klasa jest klasą publiczną,

class→rozpoczynamy budowanie klasy,

nawiasy { } po nazwie programu→ciało głównej klasy programu,

main()→metoda(funkcja), która musi zawierać klasę główną aby uruchomić program,

public (modyfikator), static(specyfikator)→oba te słowa przed main powodują, że main jest wykonywane jako pierwsza funkcja w programie,

void→oznacza, że metoda ta nie zwraca żadnych wartości,

String args[] → program będzie przekazywać w postaci tekstu argumenty funkcji do przetwarzania.

Zadanie1

Napisać program, który prezentowałby dane personalne programisty w formie wizytówki ucznia (nazwisko, imię, adres zamieszkania, zainteresowania → dane fałszywe) na ekranie monitora po uruchomieniu programu. Całe dane w ramce ze znaków *. Co najmniej sześć linijek kodu. Użyj polskich znaków na konsali.

Polskie znaki na konsoli (sposób pierwszy):

- w oknie CMD wpisz: chcp 1250
- uruchom Właściwości okna CMD i w zakładce Czcionki należy wybrać czcionkę Lucida Console.

Polskie znaki na konsoli (sposób drugi):

Najprostrzym sposobem wymuszenia generowania znaków w odpowiedniej stronie kodowej jest użycie klasy `PrintWriter`. Aby uzyskać polskie znaki na konsoli DOS-owej należy napisać:

```
try
{
    PrintWriter o = new PrintWriter( new OutputStreamWriter(System.out,"Cp852"), true);
    o.println(" Przykładowy tekst z polskimi znakami ąęńśźźół");
}
catch (Exception e)
{
    e.printStackTrace();
}
```

Dla konsoli windowsowej należałoby wstawić stronę kodową CP1250.

Blok `try {} catch` przechwytyje wyjątek `UnsupportedEncodingException`, który może zostać wyrzucony przez konstruktora obiektu `OutputStreamWriter()`.

Obiektowy język programowania

Programowanie obiektowe (ang. object-oriented programming) — paradygmat programowania, w którym programy definiuje się za pomocą obiektów — elementów łączących stan (czyli dane, nazywane najczęściej polami) i zachowanie (czyli procedury, tu: metody). Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

Podjęcie to różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów.

Zalety programowania obiektowego w Javie

Obiektowa struktura języka Java, zapewniająca:

- wysokie bezpieczeństwo ułatwiająca wykorzystanie wcześniej już stworzonych modułów,

- współpracę między programistami,
- obiektowe podejście do analizy i projektowania prowadzi do bardziej stabilnych systemów,
- podejście obiektowe do takich systemów jest bardziej naturalnym sposobem postępowania z systemami zorientowanymi na użytkownika.

Programowanie obiektowe w Javie

Programowania zorientowanego obiektowo w Javie opiera się na pojęciu klasy.

Klasa → opis obiektu lub obiektów z jednolitym zbiorem atrybutów i usług, zawierający opis tworzenia nowych obiektów w klasie inaczej to pogrupowane wszelkie dane i akcje podejmowane na tych danych.

Klasa jest modulem posiadającym nazwę i atrybuty w postaci pól danych i metod.

deklaracja klasy w Java:

a) prosty sposób

Class nazwa_klasy

```
{
    //treść klasy
}
```

b) całościowy sposób deklaracji klasy

class nazwa_klasy **extends** nazwa_klasy_przodka **implements** nazwa_interfejsu

```
{
    deklaracje pól (atrybutów);
    deklaracje metody (funkcji);
    konstruktor klasy;
    deklaracja obiektów
}
```

W treści klasy definiujemy:

- pola,
- metody,
- konstruktor klasy,
- obiekty.

Każdą klasę jest zapisywana (tworzona podczas kompilacji) w oddzielnym pliku o nazwie zgodnej z nazwą klasy oraz rozszerzeniu class. Czyli klasę Samochod zostanie zapisana w pliku Samochod.class.

W jednym pliku tekstowym programu może być zapisane wiele klas, lecz w procesie kompilacji są one zapisywane w oddzielnych plikach *.class.

Uwaga:

Definicja klasy jest jedynym sposobem zdefiniowania nowego typu danych w Javie. Posługując się pojęciami klasy, możemy w wygodny i elegancki sposób definiować różnorodne typy danych.

Modyfikator dostępu do klasy

Modyfikator dostępu do klasy to słowo poprzedzające słowo kluczowe class np. public.

W Javie istnieją różne poziomy dostępności klas i ich elementów dla innych klas. Standardowo, zdefiniowana klasa oraz jej metody i zmienne są dostępne wyłącznie dla innych klas w tym samym pakiecie. W przypadku prostych apletów oznacza to, że dostęp do klasy mają wyłącznie inne klasy znajdujące się w tym samym katalogu.

public - klasa z tym modyfikatorem jest dostępna dla każdej innej klasy. kiedy procesor VM otrzymuje polecenie uruchomienia nowego apletu, klasa tego apletu musi być publiczna. Ale już inne klasy, do których dostęp musi mieć uruchamiany aplet, nie muszą być publiczne. Metody i zmienne elementarne klas publicznych, które zostaną same zadeklarowane jako publiczne, będą dostępne dla innych klas.

private - modyfikator, który ogranicza dostęp do elementów klasy tak, aby żadne inne klasy nie miały bezpośredniego dostępu do metod i zmiennych elementarnych tej klasy.

protected - w ten sposób deklarowane są elementy chronione, dostępne dla wybranych podklas w innych pakietach, deklarowanie metod jako chronionych pozwala na ich używanie w ich własnych klasach, a także w podklasach tych klas.

private protected - modyfikator ten zachowuje się podobnie jak protected, ale elementy zadeklarowane w ten sposób nie są dostępne dla innych klas nie będących podklasami, bez względu na to, czy znajdują się one w tym samym pakiecie.

package jest domyślnym, więc nie trzeba go wpisywać, jeżeli chcemy z niego skorzystać

MODYFIKATORY DOSTĘPU				
Nazwa modyfikatora	klasa	Pakiet	podklasa	Wszystko
Public	TAK	TAK	TAK	TAK
Protected	TAK	TAK	TAK	NIE
Packane	TAK	TAK	NIE	NIE
Private	TAK	NIE	NIE	NIE

Specyfikatory

Specyfikatory to modyfikatory, ale o innym przeznaczeniu niż określenie dostępu do klas, metod, pól.

Zastosowanie specyfikatora wiąże się z statycznymi i niestatycznymi polami i metodami.

Niestatyczne wiążą się z istnieniem jakiegoś obiektu.

Składowe statyczne są deklarowane przy użyciu specyfikatora static, mogą być używane nawet wtedy, gdy nie istnieje żaden obiekt klasy.

Uwaga:

- Specyfikatory umieszczamy zawsze po modyfikatorze dostępu, np. **public static** nigdy odwrotnie
- ze statycznych metod nie wolno odwoływać się do niestatycznych składowych klasy (obiekt może nie istnieć). Możliwe są natomiast odwołania do innych statycznych składowych.

PODSTAWOWE SPECYFIKATORY	
nazwa specyfikatora	Znaczenie
Abstrakt	Umieszczenie tego specyfikatora przed nazwą <u>klasy</u> uniemożliwia utworzenie obiektu tej klasy, wymusza więc tworzenie podklas, dzięki którym będzie można utworzyć obiekty. Mówi to o tym, iż klasa ta zawiera niezdefiniowane metody, które należy zdefiniować w podklasach. Umieszczenie tego specyfikatora przed <u>metodą</u> mówi o tym, iż metoda ta nie zawiera ciała, ale tylko sygnaturę, i należy do klasy abstrakcyjnej.
Final	<ul style="list-style-type: none"> • Umieszczenie tego specyfikatora przed nazwą <u>klasy</u> uniemożliwia

	utworzenie dla niej podklas. <ul style="list-style-type: none"> • Umieszczenie tego specyfikatora przed nazwą <u>metody</u> powoduje, że nie może ona być przysyłana. • Umieszczenie tego specyfikatora przed nazwą <u>pola</u> powoduje, że pole to nie może zmieniać swojej wartości.
Static	Umieszczenie tego specyfikatora przed nazwą <u>metody</u> powoduje, że wywołanie tej metody odbywa się nie poprzez nazwę zmiennej przechowującej obiekt tego typu, ale po prostu poprzez podanie samej nazwy klasy. W podobny sposób jak do metody odwołujemy się do <u>pola</u> klasy, a więc podając tylko nazwę klasy.

Hermetyzacja (enkapsulacji lub kapsułkowania)

Wewnątrz ciała klasy znajdują się pola i metody. Część pól i metod można odpowiednio ukryć przed "zewnętrznym światem" klasy tak jak to ma miejsce z przedmiotami ze świata rzeczywistego. Czyli np. ukrywasz dla obiektów zdefiniowane i stworzonych przez inne klasy

Przykład2

Uwaga:

Niezrozumiałym elementem programu (przykładu) poniżej jest dodawanie liczb do napisów. Otóż w Javie wynikiem takiego dodawania jest przekształcenie liczby w napis, a następnie połączenie napisów.

Wpisz oraz uruchom przykład.

```
public class ProgramLicznik
{
    public static void main(String args[])
    {
        Licznik.przelicz ();
        System.out.println("licznik="+Licznik.licznik);
        Licznik.przelicz ();
        System.out.println("licznik="+Licznik.licznik);
    }
}

class Licznik
{
    public static int licznik=0;    // public-zezwoleńie dostępu do
                                   // zmiennej dla wszystkich klas
                                   // static-zmienna statyczna
                                   // nie zależy od konkretnego obiektu
    public static void przelicz()  // public-zezwoleńie dostępu do
                                   // metody dla wszystkich klas
                                   // static-metoda statyczna
                                   // nie zależy od konkretnego obiektu
                                   // void-metoda nie zwraca wartości
    {
        licznik++;
    }
}
```

Metody klasy

- a) Każda funkcja w Javie jest związana z definicją klasy (spełnia rolę jej **metody**).
- b) Metoda posiada nazwę poprzedzoną typem wartości, jaką zwraca.
- c) Metoda może mieć identyfikator dostępu np. `public` za nazwą metody stosujemy zwykle nawiasy wewnątrz, których możemy zdefiniować parametry do przekazania. Za nawiasami okrągłymi stosuje się nawisy klamrowe, pomiędzy, którymi wpisujemy kod, który metoda ma wykonać.

Definicja **metody** ma następującą składnię:

```
typ_wyniku nazwa_metody(parametry_metody)
    { //instrukcje metody }
```

Po umieszczeniu w ciele klasy deklaracja taka będzie wyglądała następująco:

```
class nazwa_klasy
{
    typ_wyniku nazwa_metody(parametry metody)
    {
        //instrukcje metody
    }
}
```

Jeśli metoda nie zwraca żadnego wyniku, jako typ wyniku należy zastosować słowo **void**, jeśli natomiast nie przyjmuje żadnych parametrów, pomiędzy nawiasami okrągłymi nie należy nic wpisywać.

Słowo **void** oznacza, że metoda ta nie zwraca żadnego wyniku, brak parametrów *pomiędzy nawiasami okrągłymi* mówi, że metoda ta nie przyjmuje żadnych parametrów.

Po utworzeniu obiektu danej klasy możemy wywołać metodę w sposób identyczny w jaki odwołujemy się do pól klasy.

```
nazwa_zmiennej .nazwa_metody(parametr metody);
```

Pola służą do przechowywania danych, czyli jest właściwością klasy i jest definiowaną za pomocą zmiennej lub stałej

Robimy to dokładnie w taki sam sposób, jak deklarowaliśmy zmienne typów prostych (np. `short`, `int`, `char`).

```
typ_zmiennej nazwa_zmiennej;
```

W ten sposób powstała zmienna, która nie zawiera żadnych danych tzn. jest pusta. Musimy utworzyć obiekt klasy i przypisać go do zmiennej.

Obiekt to samoistna część programu, która może przyjmować określone stany i ma określone zachowania, które mogą zmieniać te stany bądź przesyłać dane do innych obiektów.

Obiekt tworzymy, podając typ obiektu, czyli, do jakiej klasy należy następnie nazwę obiektu, a po znaku równości operatora new oraz nazwę konstruktora i nawisy zwykłe. Między nawiasami mogą być, lecz nie muszą parametry.

```
np.  
Fiat obiekt_fiat;  
obiekt_fiat=new Fiat(); //konstruktor jest bezargumentowy
```

lub w skrócie

```
Fiat obiekt_fiat=new Fiat(); //konstruktor jest bezargumentowy
```

wytlumaczenie: utworzony został obiekt obiekt_fiat należący do klasy Fiat z konstruktorem Fiat(), który jest bezargumentowy.

```
np.  
Figura A;  
A = new figura("kwadrat"); //konstruktor z argumentem
```

lub w skrócie:

```
Figura A = new Figura("kwadrat");
```

Przykład3

Wpisz oraz uruchom przykład.

```
public class Przyklad3  
{  
    public static void main(String args[])  
    {  
        Nazwisko naz=new Nazwisko("Kowalski");  
                                                //stworzenie obiektu naz klasy Nazwisko  
                                                //i przekazanie do tego obiektu  
                                                //ciągu znaków  
        naz.wyswietl_nazwisko();  
                                                //metoda wyświetli tekst "Kowalski"  
                                                //przekazane do konstruktora  
                                                //jako parametr  
    }  
}  
class Nazwisko                                //definicja klasy nazwisko  
{  
    String nazwisko_1;                        //implementacja pola klasy nazwisko_1  
    public Nazwisko(String nazwisko_2)        //definicja konstruktora klasy  
    {  
                                                //pobierający parametr w postaci ciągu  
                                                //znakow  
        nazwisko_1=nazwisko_2;  
    }  
}
```

```
public void wyswietl_nazwisko()
{
    System.out.println(nazwisko_1); //w metodzie wyswietl_nazwisko()
}                                     //nakazujemy wyswietlic napis Kowalski
}
```

Przykład 4

Temat: Symulacja sklepu → trzy produkty, trzech klientów. Z użyciem programowania obiektowego.

Wpisz oraz uruchom przykład.

```

class Produkt
{
    static int liczbaproduktow=0;
// zmienna statyczna nalezy do klasy nie obiektu

    int numerklienta =0;

    Produkt (String towar) //konstruktor w klasie produkt
    {
        System.out.println(" Obsluga wprowadza do sprzedazy: " +towar +" z
eksportu" );
    }

} // koniec klasy Produkt

public class Sklep
{
    public static void main (String args[])
    {

// powołanie trzech obiektów woda, chleb, masło
// z przekazaniem parametru do konstruktora

Produkt woda = new Produkt(" woda" );
Produkt chleb = new Produkt(" chleb" );
Produkt maslo = new Produkt(" masło " );

woda.numerklienta=1 ;
woda.liczbaproduktow++;
System.out.println(" klient : " +woda.numerklienta);
System.out.println(" Liczba sprzedana wody: " +woda.liczbaproduktow);
System.out.println(" Liczba sprzedana chleba: " +chleb.liczbaproduktow);
System.out.println(" Liczba sprzedana masło: " +maslo.liczbaproduktow);

chleb.numerklienta=2;
chleb.liczbaproduktow++;
System.out.println(" klient : " +chleb.numerklienta);
System.out.println(" Liczba sprzedana wody: " +woda.liczbaproduktow);
System.out.println(" Liczba sprzedana chleba: " +chleb.liczbaproduktow);
System.out.println(" Liczba sprzedana masło: " +maslo.liczbaproduktow);

maslo.numerklienta=3;
maslo.liczbaproduktow++;
System.out.println(" klient : " +maslo.numerklienta);
System.out.println(" Liczba sprzedana wody: " +woda.liczbaproduktow);
System.out.println(" Liczba sprzedana chleba: " +chleb.liczbaproduktow);
System.out.println(" Liczba sprzedana masło " +maslo.liczbaproduktow);
    }
} // koniec public klasy Sklep

```

Zadanie 4

Wykonaj dwie klasy:

1)kwiat_nazwisko_ucznia

-wykonaj pole statyczne typu naturalnego o wartości początkowej 0 nazwa pola

ilosc_zakupow_kwiatow_nazwisko_ucznia

-wykonaj konstruktor , który nazwę kwiatu (tworzonego obiektu) zapisze podczas tworzenia obiektu

2)kwiaciarnia_nazwisko_ucznia

-utwórz cztery obiekty, kwiaty

-zasymuluj zakup trzy razy każdego z tych kwiatów z zapisanie ilości zakupów

Przykład 5

Temat: Proste obliczenia z użyciem programowania obiektowego. Powołanie do życia dwóch obiektów (okręgów) i obliczenie pola i obwodu poprzez zdefiniowanie i wywołanie metod.

Wpisz oraz uruchom przykład.

```
class Okrag
{
    static int liczbaokregow=0; //pole

    Okrag (String figura) //konstruktor
    {
        System.out.println("Utworzony okrag " +figura );
    }

    void Obwod (double r)      //metoda1
    {
        final double PI=3.14159;    //deklaracja stałej
        double obw=2*PI*r;
        System.out.println("obw=" +obw+" cm");
    }

    void Pole (double r) //metoda2
    {
        final double PI=3.14159;
        double pol=PI*r*r;
        System.out.println("pol=" +pol+" cm^2");
    }
}

public class Obliczenia
{
    public static void main (String args[])
    {

        Okrag okrag1 = new Okrag(" okrag pierwszy" ); //insatlacja nowego obiektu okrag1
        okrag1.liczbaokregow++;                          //zwiększenie zawartosci pola
        System.out.println("to jest okrag= "+okrag1.liczbaokregow);
        okrag1.Obwod(3);                                  //wywołanie motedy
        okrag1.Pole(3);
        Okrag okrag2 = new Okrag(" okrag drugi" ); //insatlacja nowego obiektu okrag2
        okrag2.liczbaokregow++;
        System.out.println("to jest okrag= "+okrag2.liczbaokregow);
        okrag2.Obwod(4);
        okrag2.Pole(4);
    }
}
```


Zadanie 5

Wykonaj dwie klasy:

1)Walec_nazwisko_ucznia

-konstruktor tej klasy wypisze „Utworzono bryłę” oraz który walec (możliwe napisy walec1 lub walec2)

-wykonaj pole statyczne typu naturalnego o wartości początkowej 0 nazwa pola

ilosc_walcy_nazwisko_ucznia

-dwie metody o nazwach i parametrach:

v_nazwisko(h-wysokość, r-promień) → objętość

pc_nazwisko(h-wysokość, r-promień) → pole powierzchni całkowitej walca

2)Oblicz_nazwisko_ucznia

-utwórz dwa obiekty, walce

o nazwach: walec1_nazwisko_ucznia oraz walec2_nazwisko_ucznia

-wykonaj zliczenie utworzonych walcy oraz obliczenia objętości i pola powierzchni całkowitej dla dwóch obiektów.

Przykład ??

```
class Main
{
    public static void main (String args[])
    {
        Punkt punkt = new Punkt();
        punkt.x = 100;
        punkt.y = 100;
        System.out.println("współrzędna x = " + punkt.pobierzX());
        System.out.println("współrzędna y = " + punkt.pobierzY());
    }
}
```

Konstruktor

Klasy posiadają specjalne metody zwane konstruktorami. Są one wywoływane, gdy tworzona jest działalność klasy np. tworzenie nowego obiektu. Konstruktor posiada taką samą nazwę jak klasa i nie zwraca żadnej wartości. Konstruktorów może być wiele (zwróć uwagę, że mają tę samą nazwę) ale mogą mieć inną budowę np. mieć argumenty lub nie. Który konstruktor zostanie wywołany przy tworzeniu obiektu zależy od tego jak wywołamy np. z parametrami lub bez. Konstruktory rezerwują zasoby dla tworzonego egzemplarza(przedstawiciela) klasy. Oprócz tego mogą wykonywać różne prace wstępne, np. ustawiać dane, coś obliczać, itd. Jeśli konstruktor nie jest zadeklarowany wówczas Java dostarcza bezargumentowego konstruktora o nazwie takiej jak nazwa klasy.

Metoda będąca konstruktorem nigdy nie zwraca żadnego wyniku i musi mieć nazwę zgodną z nazwą klasy.

```
Class nazwa_klasy
{
    nazwa_klasy()
    {
```

```
//kod konstruktora
    }
}
```

Uwaga:

W klasie może istnieć więcej konstruktorów (o tej samej nazwie) różniących się liczbą lub typem argumentów.

```
class Punkt
{
    int x;
    int y;
    Punkt()
    {
        x = 1;
        y = 1;
    }
}
```

Konstruktor nie musi być bezparametrowy, może przyjmować argumenty wykorzystywane, do zainicjowania pól obiektu.

```
class nazwa_klasy
{
    nazwa_klasy(typ1 argument1, typ2 argument2, ... , typn argumentn)
    {
    }
}
```

Jeżeli konstruktor przyjmuje argumenty należy je podać i zastosować wywołanie.

```
nazwa_klasy zmienna = new nazwa_klasy(parametry konstruktora)
```

Destruktor

Destruktoem nazywamy specjalną metodę bezparametrową, która jest wywoływana zawsze automatycznie w momencie usuwania obiektu. W Java istnieje coś podobnego do destruktora. Użycie jest następujące, dodajesz do obiektu metodę (np. dispose()) i wywołujesz ją, gdy obiekt nie jest potrzebny. Tak jest to zrobione np. w przypadku klasy JFrame.

Podsumowanie → Klasy, obiekty i referencje

Klasa deklaruje nowy typ. Każdy obiekt jest określonej klasy. Do obiektu odwołujemy się zawsze za pośrednictwem referencji. W klasie mogą znajdować się definicje pól dostępnych w obiekcie oraz metod, jakie na obiekcie można wykonywać. Dodatkowo klasa może zawierać konstruktory, służące do inicjowania początkowego stanu obiektu. Aby zadeklarować pola i metody obiektu (inaczej pola i metody instancyjne) należy wyrzucić modyfikator static z definicji. Pola i metody obiektu będziemy nazywali po prostu polami i metodami.

Poliformizm

Polimorfizm czyli wielopostaciowość opisuje zdolność kodu Java (C++) do różnych zachowań zależnie od sytuacji w trakcie wykonywania programu.

Poliformizm nie jest cechą charakterystyką obiektów, jest cechą charakterystyką funkcji składowych klasy.

Implementuje się go poprzez architekturę klasy, ale tylko funkcje składowe klasy mogą być poliformiczne, nie zaś cała klasa.

Przykład ??

```
class Point {
    int x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    void print() {
        System.out.println("Point("+x+", "+y+"");
    }
}
```

Klasa Point zawiera dwa pola i jedną metodę oraz jeden konstruktor dwuargumentowy. Spójrzmy na przykład:

Przykład ??

```
class Test3 {
    public static void main(String[] args) {
        Point p = new Point(5, 3);
        p.y = 7;
        p.print();
    }
}
```

Przykład ten demonstruje konstruowanie obiektu klasy za pomocą operatora new. Operator new tworzy nowy obiekt - rezerwuje dla niego miejsce w pamięci, a następnie wykonuje konstruktor w celu zainicjowania obiektu. Wynikiem działania operatora new jest referencja (wskazanie) do stworzonego obiektu (jego adres).

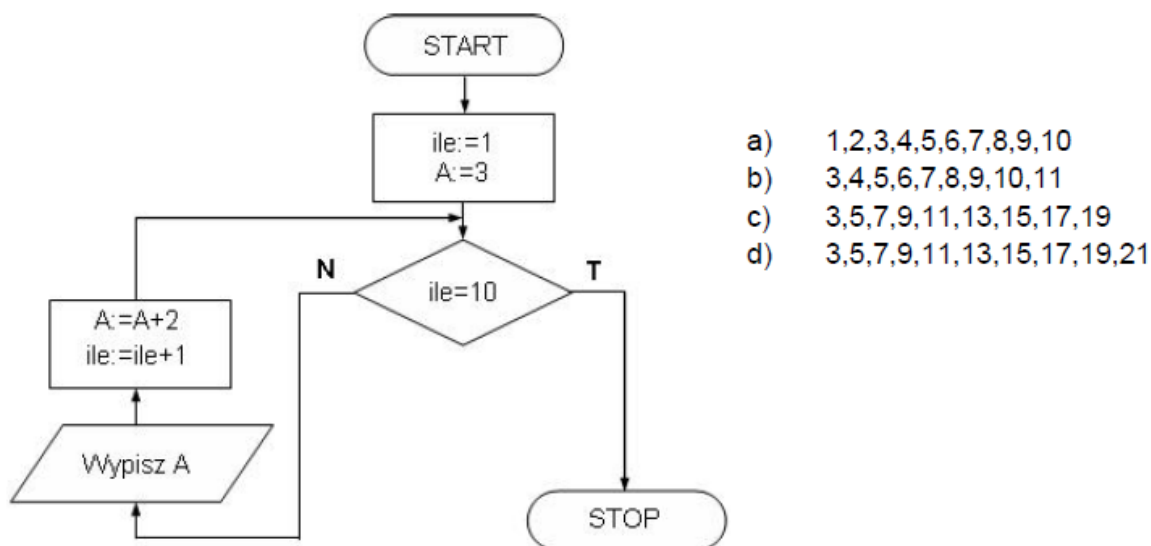
Słowo kluczowe this

Jest to odwołanie do obiektu bieżącego, możemy je traktować jako referencję do aktualnego obiektu.

Metodyk rozwiązywania zadań z programowania oraz algorytmów.

Zadanie 1

Jaki będzie ciąg liczb wyprowadzanych przez schemat blokowy.



Wykonujemy tabelę przejść:

Zawartość komórki Ile	Zawartość komórki A	Wyświetlone A	Warunek ile=10
1	3	3	1=10 → nie
.....
			10=10 → TAK

Skrócone zapisy operacji obliczeniowych dla C++ i Java

Zapis tradycyjny	Zapis skrócony pierwszy	Zapis skrócony drugi
zmienna = zmienna + zmienna1	zmienna+=zmienna1	
x = x*10	x*=10	
odp = odp +3	odp+= 3	
j = j -1	j -= 1	j-- (dwa minusy)
i = i +1	i += 1	i++ (dwa plusy)
k =k % 2	k %= 2	

Zadanie 2

Uzupełnij tabelkę

Zapis tradycyjny	Zapis skrócony pierwszy	Zapis skrócony drugi
zmienna = zmienna + zmienna1	zmienna+=zmienna1	
?	?	p++
?	zmie+= 5	
?	ujka - = 1	?
jj = jj % 2	?	
?	u*=10	
hitrus= hitrus/4	?	

Zadanie 3

Przeanalizuj działanie poniższego algorytmu dla $n=3$.

1. $s \leftarrow 1$; $p \leftarrow 1$;
2. dla $k \leftarrow 1..n$ wykonuj
3. $s \leftarrow s+p$
4. dla $i \leftarrow 1..k$ wykonuj
5. $p \leftarrow p*k$

Wstaw znak X w odpowiednim miejscu P-prawda F-falsz

	P	F
Podczas wykonywania algorytmu k dwukrotnie przyjmuje wartość 3.		
Podczas wykonywania algorytmu i dwukrotnie przyjmuje wartość 2.		
Po wykonaniu powyższego algorytmu $s=7$.		
Po wykonaniu powyższego algorytmu $p=108$.		

Rozwiązanie:

Uzupełnij tabelę

[illegible]

Zadanie 4

Ile razy zostanie wykonana pętla while po wykonaniu programu.

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    int x=5, y=10;
    while ((x!=7)||(y>16))
    {
        x++;
        y+=2;
    };
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Rozwiązanie:

a)zapisujemy w mowie potocznej warunek while czyli→

powtarzaj w pętli, inaczej wejdź do pętli (czyli to co w nawiasach { }) dopóki ((x jest różne [!=] od 7) lub (y większe od 16))

czyli dopóki warunek nie jest prawdziwy wchodzimy do pętli

b)wykonujemy tabelkę

Zmienna x	Zmienna y	Przejęcia pętli
5	10	1 zostajemy wpuszczeni do pętli bo x nie jest 7 a y jest mniejsze od 16
6	12	2 zostajemy wpuszczeni do pętli bo x nie jest 7 a y jest mniejsze od 16
7	14	Nie zostajemy wpuszczeni do pętli bo x jest 7

czyli ilość przejść 2 razy

Część 2

Obliczenia w Java w trybie konsolowym

Zmienne

Nazwa

- nie może zmienna zaczynać się od cyfry lub innych znaków niż litera,
- nie może być spacji w nazwie,
- zmienna musi mieć typ.

typy danych

typ tekstowy → String

String *zmienna_tekstowa*;

Uwaga:

- String pisane jest z dużej litery

typ znakowy → char

np.

char znak='a';

- znak musi być zapisany w pojedynczym cudzysłowie.

typ logiczny

może przyjmować tylko dwie wartości false (fałsz) lub true (prawda)

np.

boolean jest=true;

typy całkowite

a) byte

zakres od -127 do 128

np.

byte waga=82;

b) short

zakres od -32768 do 32767

np.

short lot=11182;

c) int

zakres od -2147483648 do 2147483647

np.

int wygrana=20000000;

d) long

zakres od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 808

np.

long kosmos=9 223 372 036 854 775 000

Liczby rzeczywiste (zmiennie przecinkowe)

a) float

zakres od 1.4E-45 do 3.4E+38

np.

```
float duza_liczba=3e+10;
```

b)double

zakres od 4.9E-324 do 1.7E+308

np.

```
double mam_przecinek=19.98765;
```

Uwaga:

- po przecinku stosujemy znak kropki nie przecinak

Stale

np.

```
final zmienna_tekstowa="klasa maturalna";
```

Rzutowanie zmiennych

Zmienna o typie mniejszej precyzji może być bez utraty informacji przypisana do zmiennej o typie większej precyzji.

np.

```
byte waga=100;
```

```
int waga1=waga;
```

W celu przypisania zmiennej o większej precyzji do zmiennej o mniejszej precyzji stosujemy rzutowanie zmiennych. Rzutowanie można rozpoznać po umieszczeniu w nawiasie okrągłym nowego typu.

np.

```
int waga1=120;
```

```
byte waga=(byte)waga1;
```

Zmienne statyczne i publiczne.

Dostęp do zmiennej public

Jeśli w klasie "klasa1" zadeklarujesz zmienną "zmienna" jako public, tj:

```
public int zmienna=0;
```

to w drugiej klasie możesz zrobić tak:

```
klasa1 kl = new klasa1();
```

```
System.out.println(klasa1.zmienna);
```

Dostęp do zmiennej static

Jeśli zadeklarujesz tę zmienną jako static, to będzie to zmienna statyczna (inaczej: zmienna klasy). Możesz się wtedy odwołać do niej tak:

```
klasa1.zmienna
```

czyli bez tworzenia obiektu klasy klasa1.

Jeśli nie potrzebujesz zmiennej statycznej, a odwoływał się będziesz przez konkretną instancję, to najbardziej elegancko by było gdybyś utworzył metody get i set. Jedna zwraca wartość wybranej zmiennej, a druga ją ustawia.

Zmienne statyczne nie przynależą do obiektu, przynależą one do klasy, z tego też względu powinny być wywoływane względem klasy.

Wyświetlanie liczb float, double lub BigDecimal z określoną ilością miejsc po przecinku.

np. `String.format("%.2f",obwod)`

dwa miejsca po przecinku dla zmiennej obwod.

Zadanie 6

Zapisz pytania a pod pytaniami odpowiedzi na nie.

1. Jak zbudowana może być nazwa zmiennej?
2. Opisz typ znakowy (sposób deklaracji, przykład)
3. Opisz typ tekstowy (sposób deklaracji, przykład, uwaga)
4. Opisz typ logiczny (sposób deklaracji, przykład)
5. Opisz typ całkowity (sposób deklaracji, przykład, zakres)
6. Opisz typ short (sposób deklaracji, przykład, zakres)
7. Opisz typ int (sposób deklaracji, przykład, zakres)
8. Opisz typ long (sposób deklaracji, przykład, zakres)
9. Opisz typ float (sposób deklaracji, przykład, zakres)
10. Opisz typ double (sposób deklaracji, przykład, zakres, uwaga)
11. Opisz sposób deklaracji stałych
12. Opisz rzutowanie zmiennych(opis, przykłady)
13. Opisz dostęp do zmienne statyczne z przykładem.
14. Opisz dostęp do zmienne publicznej z przykładem.
15. Wyświetlanie liczb float, double lub BigDecimal z określoną ilością miejsc po przecinku.
16. Tabela operacji arytmetycznych w Java.

Operacje na zmiennych

Arytmetyczne

Operatory arytmetyczne służą, do wykonywania operacji arytmetycznych (mnożenie, dzielenie, dodawanie, odejmowanie). Występują w tej grupie również mniej znane operatory, takie jak operator inkrementacji i dekrementacji.

Tabela: Operatory arytmetyczne w Javie

Sekwencja	Znaczenie
*	Mnożenie
\	Dzielenie
+	dodawanie
-	odejmowanie
%	dzielenie modulo (reszta z dzielenia)
++	inkrementacja (zwiększenie)
--	dekrementacja (zmniejszenie)
==	Porównanie (dwa razy równa się)
=	Przypisanie (jeden znak równa się)

Przykład 6a

Temat: Demonstracja operatorów w Java.

Wykonaj:

- Przepisz temat do zeszytu,
- Użyj nazw zmiennych:
a_trzy_litery_nazwiska.
b_trzy_litery_nazwiska.
c_trzy_litery_nazwiska.
- Dopisz obliczanie modulo a i b
- Przepisz wygląd ekranu podczas działania programu,
- Zapisz treść programu w zeszycie (z modulo, pamiętaj o nazwach zmiennych)

```
class Dodawanki_mnozonki
{
    public static void main(String args[])
    {
        int a, b, c;
        a = 20;
        b = 7;
        System.out.println("a = " + a + ", b = " + b);
        c = b - a;
        System.out.println("b - a = " + c);
        c = a / 2;
        System.out.println("a / 2 = " + c);
        c = a * b;
        System.out.println("a * b = " + c);
        c = a + b;
        System.out.println("a + b = " + c);
    }
}
```

przykładowy ekran działania programu:

a=10, b=20

b-a=10

.....
.....

Przykład 6

Temat:

Program, który obliczy pole oraz obwód trójkąta prostokątnego po podaniu:

c- przeciwprostokątna

alfa-jeden z kątów ostrych trójkąta w stopniach

Program posiada menu i masz możliwość wyboru opcji menu.

Wykonaj:

- Przepisz temat do zeszytu,
- w zeszycie rysunek trójkąta z zaznaczonym kątem i przeciwprostokątną,
- zapisz wzory, które pozwolą rozwiązać zadanie, (możesz użyć treści przykładu → patrz poniżej),
- uruchom program wpisz dane i poproś nauczyciela w celu zaliczenia.

Wy tłumaczenie działania niektórych poleceń z przykładu poniżej.

* System.in - to strumień umożliwiający dostęp do danych płynących ze standardowego wejścia (klawiatury).

* Aby móc czytać standardowe wejście linijka po linijce musimy opakować ten strumień w:

InputStreamReader - strumień przekształcający bajtowy strumień w strumień znaków.
BufferedReader - strumień buforujący dane i umożliwiający czytanie danych porcjami.

* Aby móc korzystać z klas strumieni wymienionych powyżej należy zaimportować te klasy podając pakiety, w których się znajdują. Przykładowo BufferedReader znajduje się w pakiecie java.io.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

* Metoda readLine() może spowodować wyrzucenie wyjątku IOException, który musi zostać przechwycony, lub należy dopisać do deklaracji metody throws IOException.

* Obiekty klasy String mają wiele ciekawych metod.

Dokumentacja klas standardowych Javy jest dostępna online:

<http://java.sun.com/j2se/1.5.0/docs/api/>

Treść przykładu

```
import java.io.BufferedReader; // czytanie z klawiatury
import java.io.IOException;
import java.io.InputStreamReader;
import static java.lang.Math.*; // funkcje matematyczne
```

```
public class Trojkat
{
    public static void main(String[] args) throws NumberFormatException, IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        //nowy obiekt o nazwie reader, który umożliwi wczytywanie z klawiatury
        int wariant;
        double a,b,c;
        double alfa;
        double pole;
```

```

double obwod;
System.out.println("Menu");
System.out.println("1 - obwod trojkata");
System.out.println("2 - pole trojkata: ");
System.out.print("Wprowadz wartosc 1 lub 2: wariant=");
wariant = Integer.parseInt(reader.readLine());
//konwersja z tekstu na calkowite oraz wczytywanie z kalawiatyry
System.out.print("Podaj przeciwprostokatna c=");
c = Float.parseFloat(reader.readLine());
//konwersja z tekstowego na rzeczywiste
System.out.print("Podaj jeden z katow ostrych w stopniach alfa=");
alfa = Float.parseFloat(reader.readLine());
a=c*sin(alfa*3.14/180);
b=c*cos(alfa*3.14/180);
if (wariant==1)
{
    System.out.println("Obliczam obwod trojkata");
    obwod=a+b+c;
    System.out.println("Obwod wynosi="+String.format("%.2f",obwod)+" cm");
    // %.2f oznacza dwa miejsca po przecinku
}
if (wariant==2)
{
    System.out.println("Obliczam pole trojkata");
    pole=a*b/2;
    System.out.println("Pole wynosi="+String.format("%.2f",pole)+" cm^2");
}
}
}

```

Zadanie 7

Program, który obliczy **pole całkowite oraz objętość** walca po podaniu:

D- przekątna walca,

Beta-kąt nachylenia przekątnej do płaszczyzny podstawy.

Nazwa klasy to **Walec_nazwisko_ucznia** np. Walec_kowalski (bez polskich liter)

Nazwa obiektu **czytnik_trzy_litery_nazwiska** np. czytnik_kow. W poprzednim przykładzie to reader.

Program posiada menu i masz możliwość wyboru opcji menu.

Wykonaj:

- w zeszycie rysunek walca z zaznaczeniem zmiennych,
- zapisz wzory, które pozwolą rozwiązać zadanie,
- uruchom program wpisz dane i poproś nauczyciela w celu zaliczenia,
- wykonaj trzy miejsca po przecinku z uwzględnieniem jednostek.

Dane do sprawdzenia:

D=10 alfa=45 to PC=235.562 oraz V=277,650

Dziedziczenie

- Dziedziczenie to niezwykle istotny mechanizm programowania zorientowanego obiektowo.
- Klasa pochodna to taka, która dziedziczy od klasy bazowej.
- Klasy potomne dziedziczą wszystkie składowe (pola danych i metody) swej klasy bazowej.
- Od jednej klasy bazowej może dziedziczyć wiele klas.
- W Javie nie jest możliwe wielodziedziczenie, czyli sytuacja, że jedna klasa jest pochodną dwóch klas nadrzędnych.
- Dzięki niemu mamy możliwość wielokrotnego używania tego samego kodu. Dzięki dziedziczeniu można ograniczyć się do zdefiniowania dodatkowych pól i metod, unikając przy tym wielokrotnego powtarzania fragmentów kodu.
- W Javie dziedziczenie jest wyrażane słowem `extends`.

składnia tworzenia klasy potomnej na podstawie klasy bazowej

```
class klasa_potomna extends klasa_bazowa
{
    //ciało klasy
}
```

Przykład 7

Temat: Przykład demonstruje dziedziczenie metod z klasy nadrzędnej do klasy podrzędnej.

Wykonaj:

- 1)Przepisz temat do zeszytu, łącznie z podkreśleniami.
- 2)Wpisz przykład dokonaj kompilacji, ale zmień nazwę klasy głównej na `nazwa_klasy_Samochod_nazwisko_ucznia` np. `Samochod_Kowalski`.
- 3)Wpisz przykład do zeszytu.
- 4)Zapisz w zeszycie tekst jak poniżej wraz z uzupełnieniem
NAZAWA KLASY NADRZĘDNEJ →(uzupełnij miejsce kropek)
NAZAWA KLASY PODRZĘDNEJ →(uzupełnij miejsce kropek)
SŁOWO KLUCZOWE UŻYWANE PRZY DZIEDZICZENIU TO→
- 5)Zapisz w zeszycie teksty, które wyprowadził na konsoli tekstowej program po jego uruchomieniu.

```
public class Samochod //utworzenie klasy glownej
{
    public static void main(String args[])
    {
        Fiat f=new Fiat(); // utworzenie obiektu klasy Fiat
        f.wyswietlInfo(); // wywołanie metody klasy Informacja
        f.wyswietlFiat(); // wywołanie metody klasy Fiat

        Seat s=new Seat(); // utworzenie obiektu klas Seat
        s.wyswietlInfo(); // wywołanie metody klasy Informacja
        s.wyswietlSeat(); // wywołanie metody klasy Seat
    }
}
```

```

class Informacja    //utworzenie klasy Informacja
{
    public void wyswietlInfo() //definicja metody
    {
        System.out.print("To jest samochod ");
    }
}

class Fiat extends Informacja    //utworzenie klasy Fiat
                                //jako klasy podrzędnej
                                //klasy Informacja
{
    public void wyswietlFiat() //definicja metody
    {
        System.out.println("Fiat");
    }
}

class Seat extends Informacja
{
    public void wyswietlSeat()
    {
        System.out.println("Seat");
    }
}

```

Przysłanianie metody

Przysłanianie metody zachodzi wtedy, gdy klasa podrzędna ma taką samą zdefiniowaną metodę jak jej klasa nadrzędna. Metody są identyczne wtedy, gdy mają te same sygnatury, czyli:

- te same nazwy
- parametry
- zwracany typ danych

Przykład 8

Temat: Przykład demonstruje dziedziczenie metod z klasy nadrzędnej do klasy podrzędnej oraz przesyłanie metod między tymi klasami.

Wykonaj:

- 1)Przepisz temat do zeszytu, łącznie z podkreśleniami.
- 2)Wpisz przykład dokonaj kompilacji, ale zmień nazwę klasy głównej na nazwa_klasy_Geometria_nazwisko_ucznia.
- 3)Wpisz przykład do zeszytu.
- 4)Zapisz w zeszycie teksty, które wyprowadził na konsoli tekstowej program po jego uruchomieniu.

```

public class Geometria
{
    public static void main(String args[])

```

```

{
    Figura k=new Kwadrat();
    k.wyswietlInfo();
    Figura e=new Elipsa();
    e.wyswietlInfo();
    Figura i=new Inna();
    i.wyswietlInfo();
}
}

class Figura
{
    public void wyswietlInfo()
    {
        System.out.println("To jest figura geometryczna.");
    }
}

class Kwadrat extends Figura
{
    public void wyswietlInfo() // przysłanianie metody
    {                          // wyswietlInfo() z Figura
        System.out.println("To jest kwadrat.");
    }
}

class Elipsa extends Figura
{
    public void wyswietlInfo() // przysłanianie metody
    {                          // wyswietlInfo() z Figura
        System.out.println("To jest elipsa.");
    }
}

class Inna extends Figura
{
    // brak definicji metody czyli działa metoda z Figura
    // wyswietlInfo()
}

```

Zadanie 8

Wykonaj:

- a) program, który demonstrował będzie dziedziczenie
- b) utwórz klasę Ogródek_nazwisko_ucznia np. Ogródek_Kowalski, a w niej powoła cztery obiekty warzywa na podstawie czterech klas zdefiniowanych w punkcie d), wywołaj dla każdego obiektu metodę
- c) utwórz klasę Warzywa_nazwisko_ucznia np. Warzywa_Kowalski, w tej klasie zdefiniuj metodę piszInfo_nazwisko_ucznia() piszInfo_Kowalski(), metoda ta będzie wyświetlała tekst „To jest ogródek nazwisko_ucznia” ← tutaj wpisz Twoje nazwisko.

d)utwórz cztery klasy dla czterech warzyw wybranych przez ciebie np. Pomidor_nazwisko_ucznia np. Pomidor_Kowalski na podstawie klasy Warzywa_nazwisko_ucznia np. Warzywa_Kowalski, w trzech pierwszych klasach z czterech klas warzyw zdefiniuj metodę piszInfo_nazwisko_ucznia() np. piszInfo_Kowalski(), metoda ta wyświetli napis „ to jest warzywo” np. „to jest pomidor”, czwartą klasę pozostaw bez definiowania metody.

Uruchom program, gdy działa zapisz w zeszycie.

Koniec zadania 8

Przykład bez numeru

```
class Main
{
    public static void main (String args[])
    {
        Punkt3D punkt = new Punkt3D();
        System.out.println("x = " + punkt.x);
        System.out.println("y = " + punkt.y);
        System.out.println("z = " + punkt.z);
        System.out.println("");
        punkt.ustawX(100);
        punkt.ustawY(200);
        System.out.println("x = " + punkt.x);
        System.out.println("y = " + punkt.y);
        System.out.println("z = " + punkt.z);
        System.out.println("");
        punkt.ustawXY(300, 400);
        System.out.println("x = " + punkt.x);
        System.out.println("y = " + punkt.y);
        System.out.println("z = " + punkt.z);
        System.out.println("");
    }
}
```

Instrukcja warunkowa IF

budowa instrukcji IF

```
if (warunek) {operacja, jeśli warunek został spełniony;}
```

Instrukcja wyboru IF.....ELSE

budowa instrukcji IF.....ELSE

```
if (warunek) {operacja, jeśli warunek został spełniony;}
                                     else
                                     {operacja, jeśli warunek nie został spełniony;}
```

Instrukcja switch

Instrukcja switch pozwala w wygodny sposób sprawdzić ciąg warunków i wykonywać różny kod w zależności od tego, czy są one prawdziwe czy fałszywe. W postaci ogólnej instrukcja wygląda następująco:

```
switch(wyrażenie_zmienna)
{
    case wartość1:
        Instrukcje1;
        break;
    case wartość2:
        Instrukcje2;
        break;
    case wartość3:
        Instrukcje3;
        break;
    default:
        Instrukcje4;
}
```

Należy ją rozumieć jako: sprawdź wartość wyrażenia wyrażene. Jeśli to wartość1, wykonaj instrukcje1 i przerwij wykonywanie bloku switch (instrukcja break). Jeśli jest to wartość2, wykonaj instrukcje2 i przerwij wykonywanie bloku switch, jeśli jest to wartość3, wykonaj instrukcje3 i przerwij wykonywanie bloku switch. Jeśli nie zachodzi żaden z przypadków, wykonaj instrukcje4 i zakończ blok switch.

Przykład 8a (aby rozwiązać zadanie 9)

Temat Demonstracja użycia instrukcji case.

Treść: Zmiennej *liczba* nadajemy wartość początkową 25. Instrukcją *case...switch* rozpoznajemy czy liczba jest 25 lub 15. Gdy liczba nie jest 15 lub 25 to komputer wypisuje napis "Zmienna liczba nie jest równa ani 15, ani 25."

Wykonaj:

a)Użyj nazw zmiennych:

liczba_trzy_litery_nazwiska.

b)Wpisz do zeszytu składnie instrukcji

-switch

c)Dokonaj kopiowania programu z instrukcji i jego uruchomienia.

d)dokonaj zmian w przykładzie tak aby komputer pytał się o liczbę i:

- gdy wczytasz numer w dzienniku to napisal „ to jest twój numer z dziennika”

- gdy wczytasz liczbę 100+numer_w_dzienniku to napisze „ to jest twój numer z dziennika +100”

-gdy nie jest to liczba numer w dzienniku oraz 100+numer w dzienniku to napisze „ bardzo źle nie znasz mojego numeru w dzienniku” (użyj default).

e)przepisz poprawiony przykład do zeszytu.

```

class Main
{
    public static void main (String args[])
    {
        int liczba = 25;
        switch(liczba)
        {
            case 25 :
                System.out.println("liczba = 25");
                break;
            case 15 :
                System.out.println("liczba = 15");
                break;
            default :
                System.out.println("Zmienna liczba nie jest równa ani 15, ani 25.");
        }
    }
}

```

Zadanie 9

a) Wpisz do zeszytu składnię instrukcji

-IF

-IF ELSE

Uwaga:

Użyj nazw zmiennych:

a_trzy_litery_nazwiska.

b) Treść zadania

Napisz program, który pobierze z klawiatury rok, miesiąc oraz dzień, aby zwrócić dzień tygodnia w jakim wypada wprowadzony dzień (Poniedziałek, Wtorek.....). Na podstawie algorytm Zellera (opis patrz poniżej).

Do wyboru dnia na podstawie wartości **zmiennej dzien_tygodnia_trzy_litery_nazwiska** użyj instrukcji switch.

ALGORYTM

Wieczny kalendarz jest to algorytm, który na podstawie daty wyznacza dzień tygodnia wg kalendarza juliańskiego (czyli obowiązującego w całym świecie chrześcijańskim).

Algorytm Zellera został uproszczony przez matematyka, Mike'a Keitha do postaci:

Gdzie:

y-rok

m-miesiąc

d-dzień

1) oblicz **z**

$z = y - 1$ jeśli $m < 3$,

$z = y$ w pozostałych przypadkach

2) oblicz **dzień_tygodnia**

$$x = ([23 * m / 9] + d + 4 + y + [z / 4] - [z / 100] + [z / 400])$$

gdy $m \geq 3$

$$\text{dzień_tygodnia} = (x - 2) \bmod 7$$

gdy $m < 3$

$$\text{dzień_tygodnia} = x \bmod 7$$

gdzie:

[] oznacza dzielenie bez reszty z zaokrągleniem w dół

mod - funkcja modulo → przypomnij sobie jaką jest używana w Java, ale nie mod

m - numer miesiąca (od stycznia = 1 do grudnia = 12)

d - numer dnia miesiąca

y - numer roku

3)

jeżeli **dzień_tygodnia** jest równe 0 badany dzień wypada w Niedzielę, 1 w Poniedziałek etc.
Wypisz dzień tygodnia na ekran.

Uwagi do użycia algorytmu w programie:

1) wczytaj dane numer dnia, numer miesiąca, numer roku

2) oblicz **z** na podstawie wartości **m** i **y**, użyj instrukcji **IFELSE.....**

3) oblicz **x** na podstawie wartości **m, z, d, y**

4) oblicz **dzień_tygodnia** na podstawie **x** i **m**, użyj instrukcji **IFELSE.....**

5) ustal jaki jest dzień tygodnia na podstawie wartości zmiennej **dzień_tygodnia** użyj instrukcji **switch**,

6) wyprowadź jak jest dzień tygodnia dla podanej daty.

Uwagi do pisania programu w Java.

W javie dzielenie całkowite realizowane jest przez zwykły operator dzielenia „/” tylko liczby które dzielimy jak i zmienna do której zapisujemy wynik musi być typu int.

Wszystkie zmienne zadeklaruj jako int. Podczas dzielenia z użyciem operatora „/” konieczna jest konwersja na typ całkowity, który można zrealizować jak przykład poniżej.

$$x = [23 * m / 9] \quad \rightarrow \quad x = (\text{int})((23 * \text{miesiac}) / 9)$$

Zadanie 10

Uwaga:

Użyj nazw zmiennych:

a_trzy_litery_nazwiska.

treść zadania

Rozwiąż równanie kwadratowe, wg listy kroków.

krok1: wczytaj A, B, C

krok2: czy A=0 jeśli tak idź do kroku 10 jeśli nie przejdź dalej

krok3: oblicz deltę $\Delta = B^2 - 4 \cdot A \cdot C$

krok4: czy $\Delta = 0$ jeśli tak to idź do kroku7

krok5: czy $\Delta > 0$ jeśli tak to idź do kroku8

krok6: czy $\Delta < 0$ jeśli tak to idź do kroku9

krok7: oblicz $x_0 = -B/(2A)$, wyświetl x_0 , idź do kroku 11

krok8: oblicz $x_1 = (-B + \sqrt{\Delta})/(2A)$, $x_2 = (-B - \sqrt{\Delta})/(2A)$, wyświetl x_1 , x_2 , idź do kroku 11

krok9: wyświetl „równanie nie ma rozwiązania”, idź do kroku 11

krok10: wyświetl „to nie jest równanie kwadratowe”

krok11: koniec

Dane do testowania:

Współczynnik a	Współczynnik b	Współczynnik c	komunikat
0	1	3	To nie jest równanie kwadratowe
1	1	1	Brak rozwiązań
1	4	4	Jedno rozwiązanie $x_0 = -2$
1	1	-2	Dwa rozwiązania $x_1 = -2$ $x_2 = 1$

koniec zadania

Pętla while

Pętla typu while służy, do wykonywania powtarzających się czynności. Ogólna postać pętli while wygląda następująco:

While (wyrażenie warunkowe)

```
{  
    Instrukcje;  
}
```

Instrukcje są wykonywane tak długo, dopóki wyrażenie warunkowe jest prawdziwe.

Jest instrukcja pętli z **kontrolowanym wejściem**.

Zobaczmy zatem, jak za pomocą pętli while wyświetlić 10 razy napis.

Przykład do wpisania do zeszytu

```
class Main  
{  
    public static void main (String args[])  
    {  
        int i = 0;  
        while(i < 10)  
        {  
            System.out.println("Pętla w Javie");  
            i++;  
        }  
    }  
}
```

```
}  
}
```

Pętla do...while

Odmianą pętli while jest pętla do..while, której ogólna postać wygląda następująco:

```
do  
{  
    instrukcje;  
}  
while(warunek);
```

Konstrukcję tę należy zrozumieć następująco: wykonuj instrukcje dopóki warunek jest prawdziwy gdy jest nieprawdziwy to wychodzimy z petli.
Jest instrukcja pętli z **kontrolowanym wyjściem**.

Przykład do wpisania do zeszytu

```
class Main  
{  
    public static void main (String args[])  
    {  
        int i = 0;  
        do  
        {  
            System.out.println("[i = " + i + "] Pętla w Javie");  
        }  
        while(i++ < 9);  
    }  
}
```

Logiczne

Operacje logiczne możemy wykonywać na argumentach, które posiadają wartość logiczną: prawda lub fałsz. W językach programowania wartości te są oznaczone jako true i false. W przypadku Javy oba te słowa muszą być zapisane małymi literami.

Tabela Operatory logiczne w Javie

Operator	Symbol
AND	& &
OR	
NOT	!

a)Logiczny iloczyn

Wynikiem operacji AND (iloczyn logiczny) jest wartość true, wtedy i tylko wtedy, kiedy oba argumenty mają wartość true. W każdym innym przypadku wynikiem jest false.

b)Logiczna suma

Wynikiem operacji OR (suma logiczna) jest wartość false, wtedy i tylko wtedy, kiedy oba argumenty mają wartość false. W każdym innym przypadku wynikiem jest true.

c) Logiczna negacja

Operacja NOT (logiczna negacja) zamienia wartość argumentu na przeciwną. Czyli jeśli np. argument będzie miał wartość true, będzie miał wartość false i odwrotnie.

Przykład 8b

Temat: Demonstracja sprawdzania warunków z użyciem pętli **do while** oraz **while**.

Czyli:

- prawidłowy wybór płci kobieta (K/k) mężczyzna (M/m),
- wczytanie wieku, wiek z przedziału <20,70>,
- wczytanie wagi, waga z przedziału (1,20> lub (90,100>,
- jak skończyć program na klawisz „t” lub „T” na każdy inny klawisz powtórzymy program.

Wykonaj:

- 1) przepisz temat (łącznie z podpunktami)
- 2) Opis w zeszycie instrukcje (opis teoretyczny oraz przykład do zapisania w zeszycie) oraz jak nazywana jest każda instrukcja
 - a) Pętla do...while
 - b) Pętla while
 - c) tabelę Operatory logiczne w Javie.
- 3) przeczytaj uwagę tuż przed treścią przykładu i znajdź znaki ! w treści przykładu.
- 4) Wykonaj testowanie programu dla wartości jak w tabeli poniżej:

Zmienna	Wartość
plec	P
plec	K
wiek	10
wiek	90
wiek	20
waga	0
waga	50
waga	120
waga	95
dalej	N
plec	M
wiek	-5
wiek	70
waga	-4
waga	11
dalej	T

Uwaga:

W Pascalu występowała pętla **repeat until** *warunek*; która działała aż do poprawności warunku w Java **do while** *warunek* działa aż do niepoprawności warunku i dlatego wygodniej jest zapisywać warunek taki jak w pętli repeat until z Pascala i zaprzeczać poprzez użycie ! (znak wykrzyknik to zaprzeczenie w Java). Zwróć na to uwagę w przykładzie.

Treść przykładu:

```
import java.io.BufferedReader;// czytanie z klawiatury
import java.io.IOException;
```

```

import java.io.InputStreamReader;
import static java.lang.Math.*; // funkcje matematyczne

public class Main
{
    public static void main(String args[]) throws NumberFormatException, IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        //nowy obiekt, ktory umozliwi wczytywanie z klawiatury
        int wzrost;
        int wiek;
        int waga;
        String plec,dalej;
        do // poczatek petli powtarzania programu
        {
            do
            {
                System.out.println("Bede tak dlugo wczytywal kod plci, az bedzie to kobieta lub mezczyzna");
                System.out.println("Podaj plec mezczyzna lub kobieta wpisz: [M(m) lub K(k)]=");
                plec = reader.readLine();
            }
            while ((!plec.equals("m"))&&!plec.equals("M"))&&!plec.equals("k")&&!plec.equals("K"));

            System.out.println("wczytana plec="+plec);

            do
            {
                System.out.print("Bede powtarzal tak dlugo,az wczytasz wiek z przedzialu <20,70> ");
                System.out.print("\nPodaj wiek=");
                wiek = Integer.parseInt(reader.readLine());
            }
            while (!(wiek>=20)&&(wiek<=70));
            // zwroc uwage na wykrzyknik w lini powyzej

            System.out.println("wczytana wiek="+wiek);

            do
            {
                System.out.print("Bede powtarzal tak dlugo,az wczytasz wage z przedzialu (1,20> lub (90,100> ");
                System.out.print("Podaj wage=");
                waga = Integer.parseInt(reader.readLine());
            }
            while ( ! ( ( (waga>1)&&(waga<=20) )||( (waga>90)&&(waga<=100) ) ) );
            // zwroc uwage na wykrzyknik w lini powyzej
            System.out.println("wczytana waga="+waga);

            System.out.println("Czy chcesz skonczyc program ? nacisnij T(t)=");
            System.out.println("kazdy inny klawisz powtorzy program");
            dalej = reader.readLine();
        }
        while ((!dalej.equals("T"))&&!dalej.equals("t")); // koniec petli powtarzania programu
    }
}

```

Przykład (nieoceniany)

Przykład pokazuje jak skończyć powtarzanie programu (w tym przypadku jest to napis „Lubię programować w Java”) na klawisz „k” lub „K” na każdy inny klawisz powtórzymy program.


```

import java.io.BufferedReader;// czytanie z klawiatury
import java.io.IOException;
import java.io.InputStreamReader;

public class Main
{
    public static void main(String args[]) throws NumberFormatException, IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        //nowy obiekt, który umożliwi wczytywanie z klawiatury
        String dalej;
        do
        {

            System.out.println("Lubię programować w Java");
            System.out.println("Jeśli chcesz skończyć wyświetlać napis naciśnij K(k)=");
            System.out.println("każdy inny klawisz powtórzy napis");
            dalej = reader.readLine();
        }
        while ((!dalej.equals("K"))&&(!dalej.equals("k")));

    }
}

```

Zadanie 10a

Przeczytaj całą treść zadania(aż do napisu_Koniec zadania 10a)

Treść zadania

Napisz program, który będzie obliczał po podaniu boków trójkąta:

- pole trójkąta,
- promień okręgu opisanego na trójkącie,
- promień okręgu wpisanego w trójkąt.

Program będzie posiadał menu, trzy opcje. Program będzie sprawdzał poprawność wczytywania danych. Program będzie miał możliwość powtarzania na wciśnięcie klawisza „P” lub „p”. Program zapyta się czy chcesz powtórzyć i gdy wciśniemy „P” lub „p” nastąpi powtórzenie obliczeń czyli wyświetlonę zostanie menu oraz pytanie o opcje.

Oblicz:

Opcja 1:

pole trójkąta po podaniu boków trójkąta a_trzy_litery_nazwiska_ucznia,
b_trzy_litery_nazwiska_ucznia, c_trzy_litery_nazwiska_ucznia, korzystając z wzoru Herona

$$t = (a + b + c) / 2$$

$$P = \sqrt{t * (t - a) * (t - b) * (t - c)}$$

Opcja 2:

promień r_op_trzy_litery_nazwiska_ucznia →okręgu opisanego na trójkącie po podaniu boków a_trzy_litery_nazwiska_ucznia, b_trzy_litery_nazwiska_ucznia,
c_trzy_litery_nazwiska_ucznia,

$$t = (a + b + c) / 2$$

$$P = \sqrt{t * (t - a) * (t - b) * (t - c)}$$

$$r_op = (a * b * c) / (4 * P)$$

Opcja 3:

promień `r_wp_trzy_litery_nazwiska_ucznia` → okręgu wpisanego w trójkąt po podaniu boków `a_trzy_litery_nazwiska_ucznia`, `b_trzy_litery_nazwiska_ucznia`, `c_trzy_litery_nazwiska_ucznia`,

$$t = (a + b + c) / 2$$

$$P = \sqrt{t * (t - a) * (t - b) * (t - c)}$$

$$r_wp = (2 * P) / (a + b + c)$$

Zadanie rozwiązuje etapami:

1)zapewnij wczytywanie **`a_trzy_litery_nazwiska_ucznia`**, **`b_trzy_litery_nazwiska_ucznia`**, **`c_trzy_litery_nazwiska_ucznia`** jak liczby rzeczywiste →sprawdź działanie programu.

2)zapewnij wczytywanie **`opcja_trzy_litery_nazwiska_ucznia`**, jako liczba naturalna(całkowita) →sprawdź działanie programu.

3)Sprawdź pętlą *Do while* czy zmienna **`opcja_trzy_litery_nazwiska_ucznia`** przyjmuje tylko trzy wartości {1, 2, 3}. Wartości 1,2,3 wynikają tego, że są trzy opcje do wyboru `opcja1`, `opcja2`, `opcja3`→sprawdź działanie programu.

4) Wykonaj wejście do opcji programu z użyciem instrukcji **if** np.

po wyborze opcji1 zobaczymy napis” jesteś w opcji 1”

po wyborze opcji2 zobaczymy napis” jesteś w opcji 2”

po wyborze opcji3 zobaczymy napis” jesteś w opcji 3”

→sprawdź działanie programu.

5)Oblicz pole oraz wyprowadź jego wartość na ekran, zastosuj dwa miejsca po przecinku. Dla wartości zmiennych `a_trzy_litery_nazwiska_ucznia`, `b_trzy_litery_nazwiska_ucznia`, `c_trzy_litery_nazwiska_ucznia`. Dla `a=3`, `b=4`, `c=5` pole wyjdzie 6. →sprawdź działanie programu.

6)Oblicz promienia opisanego na trójkącie (pamiętaj, że aby obliczyć ten promień najpierw musisz obliczyć `t` i `P`) oraz wyprowadź jego wartość na ekran, zastosuj dwa miejsca po przecinku. Dla wartości zmiennych `a_trzy_litery_nazwiska_ucznia`, `b_trzy_litery_nazwiska_ucznia`, `c_trzy_litery_nazwiska_ucznia`. Dla `a=3`, `b=4`, `c=5` promień okręgu opisanego wyjdzie 2.5. →sprawdź działanie programu.

7)Oblicz promienia wpisanego w trójkąt (pamiętaj, że aby obliczyć ten promień najpierw musisz obliczyć `t` i `P`) oraz wyprowadź jego wartość na ekran, zastosuj dwa miejsca po przecinku. Dla wartości zmiennych `a_trzy_litery_nazwiska_ucznia`, `b_trzy_litery_nazwiska_ucznia`, `c_trzy_litery_nazwiska_ucznia`. Dla `a=3`, `b=4`, `c=5` promień okręgu wpisanego wyjdzie 1.0. →sprawdź działanie programu.

8)Sprawdź pętlą *Do while* czy zmienne są `a_trzy_litery_nazwiska_ucznia`, `b_trzy_litery_nazwiska_ucznia`, `c_trzy_litery_nazwiska_ucznia` są większe od zera (boki trójkąta muszą być liczbami dodatnimi). Wykonaj trzy pętle, dla każdej zmiennej osobno.

9)opisz jedną pętlę *Do while* obejmującą pętlę z punktu 8) tak aby sprawdzić warunek zamykania trójkąta. Jaki to warunek pomyśl sam.

10) opisz jedną pętlę *Do while* obejmującą cały program tak aby program powtarzał się na klawisze „P” lub „p” na inne kończył działanie(patrz przykład 8c).

Koniec zadania 10a

Losowanie

Pakiet

W Javie do losowania służy pakiet:

java.util.Random.

Metody

Zawiera on metody, które umożliwiają wygenerowanie liczb pseudolosowych.

a)Metoda nextInt()

b)oraz jej przeciążony odpowiednik nextInt(int n) służąca do generowania liczb naturalnych.

np.

```
import java.util.Random;
```

```
// jakaś część programu
```

```
    Random losuj = new Random();
```

```
    int liczba;
```

```
    liczba = losuj.nextInt();
```

```
//jeśli chcemy ograniczyć przedział losowanych liczb do zadanej wartości, podajemy tę
```

```
//wartość jako parametr metody nextInt() tutaj to liczba 200
```

```
// jakaś część programu
```

```
    liczba = losuj.nextInt(200);
```

```
//możemy również wylosować liczby typu int z dowolnie wybranego przedziału: <MIN,
```

```
MAX) - przedział prawostronnie otwarty:
```

```
// jakaś część programu
```

```
    liczba = losuj.nextInt(MAX - MIN) + MIN;
```

Przykład 9

a)Wpisz do zeszytu

-jaki pakiet umożliwia losowanie

-w jaki sposób losujemy liczby zadanego przedziału

Przykład pyta się o dwie liczby:

min→ dolna wartość przedziału losowania

max→ górna wartość przedziału losowania

i losuje z tego przedziału <min,max) 3 liczby

```
import java.io.BufferedReader;
```

```
import java.io.IOException;
```

```
import java.io.InputStreamReader;
```

```
import java.util.Random; // pakiet losowania
```

```
public class Losowanie
```

```
{
```

```
    public static void main(String[] args) throws NumberFormatException, IOException
```

```
    {
```

```
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
```

```
        Random generator = new Random();
```

```
        int liczba1, liczba2, liczba3;
```

```

        int max, min;
        System.out.print("Wprowadź górną wartość przedziału losowania: ");
        max = Integer.parseInt(reader.readLine());
        System.out.print("Wprowadź dolną wartość przedziału losowania: ");
        min = Integer.parseInt(reader.readLine());
        liczba1 = generator.nextInt(max - min) + min;
        System.out.println("wylosowana pierwsza liczba"+liczba1);
        liczba2 = generator.nextInt(max - min) + min;
        System.out.println("wylosowana druga liczba"+liczba2);
        liczba3 = generator.nextInt(max - min) + min;
        System.out.println("wylosowana trzecia liczba"+liczba3);
    }
}

```

Koniec przykładu 9.

Zadanie 11

Uwaga:

Użyj nazw zmiennych:

a_trzy_litery_nazwiska.

Gra "większe-mniejsze". Zasady gry:

Celem użytkownika jest odgadnięcie liczby, wylosowanej przez komputer. Liczba jest zapamiętana i ukryta przed użytkownikiem. Użytkownik podaje kolejne liczby, natomiast komputer w zależności od tego czy podana liczba jest mniejsza czy większa od wylosowanej zwraca komunikat "Za dużo" lub "Za mało". Gra kończy się, gdy użytkownik prawidłowo odgadł wartość liczby. Prowadzony jest licznik, który pokazuje ile razy użytkownik zgadywał. Liczba losowana jest z przedziału <numer_w_dzienniku, numer_w_dzienniku+50>. Program wyświetli na początku działania informację, z jakiego przedziału wylosował liczbę.

Przykładowy wygląd ekranu:

Losuję liczbę z przedziału <5,55>

Zgadujesz 1 raz

Podaj liczbę= 6

Za mało

Zgadujesz 2 raz

Podaj liczbę= 46

Za dużo

.....

Uwaga: do powtarzania użyj pętla do...while lub while warunek zakończenia to, że liczba losowana jest równa liczbie zgadywanej, do zliczania utwórz zmienną licznik_nazwisko i dokonuj inkrementacji tej zmiennej po przejściu pętli

Część 3

Pętle i tablice, operacje bitowe, wyjątki

Pętla for

Budowa:

```
for (wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące)
{
    Instrukcja do wykonania
}
```

wyrażenia początkowe jest stosowane do zainicjowania zmiennej używanej jako licznik ilości wykonań pętli.

Wyrażenie warunkowe określa warunek, jaki musi być spełniony, aby dokonać kolejnego przejścia w pętli,

wyrażenie modyfikujące jest zwykle używane do modyfikacji zmiennej będącej licznikiem.

Przykład (niepunktowany)

```
class Main
{
    public static void main (String args[])
    {
        for(int i = 0; i < 10; i++)
        {
            System.out.println("Pętla w Javie");
        }
    }
}
```

Taką konstrukcję należy rozumieć następująco: zadeklaruj zmienną i przypisz jej wartość zero (int i = 0), następnie, tak długo dopóki wartość i jest mniejsza od 10, wykonuj instrukcje System.out.println("Pętla w Javie") oraz zwiększaj i o jeden. Tym samym na ekranie pojawi się 10 razy napis Pętla w Javie

Zadanie 12

Temat: Użycie pętli **for** do znajdowania liczb pierwszych ze wzoru Eulera.

Wykonaj:

1)Przepisz temat.

2)Zapisz w zeszycie **składnię** instrukcji FOR. Uwzględnij: wyrażenia początkowe, wyrażenie warunkowe, wyrażenie modyfikujące. Przepisz przykładu oraz treść pod nim

3)Uwzględnij uwagę podczas rozwiązywania zadania.

4)Po rozwiązaniu zadania zapisz do zeszytu 7 liczb pierwszych wypisywanych przez ten program.

Uwaga:

Użyj nazw zmiennych:

a_trzy_liter_nazwiska.

Początek zadania

- **Liczba pierwsza**, liczba naturalna $n > 1$, dla której istnieją tylko dwa dzielniki naturalne: 1 i n . Największą znaną liczbą pierwszą jest $2^{69725932}-1$ (7 VII 1999), liczba ta zapisana w systemie dziesiętnym składa się z ponad 2 mln cyfr. W roku 1772 Leonard Euler podał wzór wielomianu o postaci: $w(i) = i^2 + i + 41$ którego wartości dla $i = 0, 1, 2, \dots, 39$ są liczbami pierwszymi.
- Napisz program znajdujący czterdzieści liczb pierwszych z użyciem pętli, korzystając z wzoru podanego przez Eulera.

Wygląd ekranu:

Liczba o numerze 0 ma wartość 41

Liczba o numerze 1 ma wartość 43

.....
.....

Koniec zadania 12

Zadanie 12a

Temat: Użycie pętli **for** do rysowania w trybie tekstowym

Z użyciem 3 trzech pętli narysuj prostokąt z gwiazdek o zadanych (obliczonych) wymiarach a i b .

```

                b
*****
*                               *
*                               *      a
*                               *
*                               *
*****
```

Program będzie obliczał (z użyciem odpowiednich formuł) a i b ze wzorów zapisanych poniżej:

$a = [(\text{liczba_liter_nazwiska}) \bmod 4] + 5$

$b = [(\text{liczba_liter_imienia}) \bmod 6] + 20$

Uwaga: Pierwsza osoba (oraz zadania indywidualne), które rozwiążą zadania z użyciem będą punktowane +1 punkt.

Koniec zadania 12a

Inkrementacja i dekrementacja

Operator inkrementacji, czyli zwiększenia ($++$) zwiększa wartość zmiennej o jeden, a operator dekrementacji ($--$) zmniejsza wartość zmiennej o jeden.

Mogą występować one w formie przedrostkowej lub przyrostkowej.

Przykładowo: jeśli mamy zmienną o nazwie x, forma *przedrostkowa* będzie miała postać ++x, natomiast forma *przyrostkowa*, postać x++.

Przykład 10

Temat: Demonstracja inkrementacji oraz dekrementacji.

Wykonaj:

- 1) Zapisz w zeszycie temat.
- 2) Zapisz w zeszycie co to jest inkrementacja oraz dekrementacja uwzględnieniem formy *przedrostkowa* oraz formy *przyrostkowa*.
- 3) Dokonaj kopiowania przykładu oraz zmień:
 - Użyj nazw zmiennych (czyli zmień w przykładzie): a_trzy_litery_nazwiska.
 - nazwa klasy nazwisko_ucznia_p10 np. Kowalski_p10,
- 4) Uruchom przykład ze zmienionymi nazwami zmiennych i zapisz w zeszycie wynik działania tego przykładu czyli to co ukazuje się na ekranie.
Wynikiem działania programu będzie ciąg liczb
Zamiast kropek wpisz ciąg liczb w zeszycie.
- 5) Na podstawie opisu „Wy tłumaczenie działa programu.” (patrz pod treścią przykładu), zapisz w zeszycie jak jest różniaca ++i a i++.

```
class Main // zmiana nazwy klasy przez ucznia
{
    public static void main(String args[])
    {
        /*1*/ int x = 3, y;
        /*2*/ System.out.println (x++);
        /*3*/ System.out.println (++x);
        /*4*/ System.out.println (x);
        /*5*/ y = x++;
        /*6*/ System.out.println (y);
        /*7*/ y = ++x;
        /*8*/ System.out.println (y);
        /*9*/ System.out.println (++y);
    }
}
```

Wy tłumaczenie działa programu.

W linii 1, deklarujemy zmienne x i y oraz przypisujemy zmiennej x wartość 3. w linii 2, stosujemy formę przyrostkową operatora ++, zatem najpierw wyświetlamy wartość zmiennej x (x = 3) na ekranie, a dopiero potem zwiększamy jej wartość o jeden (x = 4). W linii 3, postępujemy odwrotnie, to znaczy, przez stosowanie formy przedrostkowej, najpierw zwiększamy wartość zmiennej x o jeden (x = 5), a dopiero wyświetlamy tę wartość na ekranie. W linii 4, jedyną operacją jest wyświetlenie wartości x (x = 5). W linii 5, najpierw przypisujemy aktualną wartość x (x = 5) zmiennej y (y = 5) i dopiero potem zwiększamy x o jeden (x = 6). W linii 6, wyświetlamy wartość y. W linii 7, najpierw zwiększamy x o jeden (x = 7), a następnie przypisujemy tę wartość y (y = 7). W linii 9 najpierw zwiększamy y o jeden, a następnie wyświetlamy wartość na ekranie.

Zadanie 13

Temat: Proste użycie inkrementacji i dekrementacji.

Wykonaj:

1) nazwa klasy nazwisko_ucznia_z13 np. Kowalski_z13,

2) Wykonaj program, który wyświetli ciąg liczb (bez użycia pętli).

nr_w_dzienniku+30

nr_w_dzienniku+33

nr_w_dzienniku+31

nr_w_dzienniku+30

nr_w_dzienniku+32

nr_w_dzienniku+34

użyj dekrementacji lub inkrementacji zmiennej o nazwie zmiennych p_numer_w_dzienniku .

Operator dekrementacji (--) działa analogicznie jak ++, z tą różnicą, że zmniejsza wartość o jeden.

Przykład (potrzebny aby rozwiązać zadanie 13)

```
class Main
{
    public static void main(String args[])
    {
        int x = 3, y;
        System.out.println (x--);
        System.out.println (--x);
        System.out.println (x);
        y = x--;
        System.out.println (y);
        y = --x;
        System.out.println (y);
        System.out.println (--y);
    }
}
```

koniec zadania 13

Operacje Bitowe

Operatory bitowe pozwalają na wykonywanie operacji na poszczególnych bitach liczb.

Są to operacje:

- AND,
- OR,
- NOT
- XOR
- operacje przesunięcia bitów.

Są one zestawione w tabeli.

Tabela Operatory bitowe w Javie

Symbol	Opis	Przykład
~	NOT - negacja=zmiana każdego bitu	byte x=100; x=~x; (x==-101)
&	AND - bitowy iloczyn logiczny	byte a=127; b=-128; byte c=a&b; (c==0)
&=	bitowy iloczyn logiczny z przypisaniem	
	OR – bitowa suma logiczna	byte a=127; b=-128; byte c=a b; (c==-1)
=	bitowa suma logiczna z przypisaniem	
^	XOR - bitowa różnica symetryczna	byte a=100; b=12; byte c=a^b; (c==104);
^=	bitowa różnica symetryczna z przypisaniem	
>>	przesunięcie w prawo z powieleniem znaku	int a=35; a=a>>2; (a==8)
>>=	przesunięcie w prawo z powieleniem znaku i z przypisaniem	
>>>	przesunięcie w prawo bez powielenia znaku	
>>>=	przesunięcie w prawo bez powielenia znaku z przypisaniem	
<<	przesunięcie w lewo	byte a=-128; int b=a<<8; (b==-32768)
<<=	przesunięcie w lewo z przypisaniem	int a=256; a<<=30; (a==0)

Teoria poniższa wykonana na podstawie:

<http://www.algorytm.org/kurs-algorytmiki/operacje-bitowe.html>

- **Operacja bitowa "i" (ang. bitwise and)**

Jest to operacja dwuargumentowa. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

a and b

a & b

Wynikowy bit jest ustawiany na 1 tylko wówczas gdy obydwa bity argumentów ustawione są na 1.

Możemy zatem powiedzieć, że wynikiem jest 1 gdy bit a i bit b są ustawione na 1.

Tabela wyników dla tego działania jest następująca:

a		
AND	b=0	b=1
b		
a=0	0	0
a=1	0	1

- *Przykład:*
0101 and 1100 = 0100

- **Operacja bitowa "lub"** (*ang. bitwise or*)

Jest to operacja dwuargumentowa. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

a or b

a | b

Wynikowy bit jest ustawiany na 1 tylko wówczas gdy przynajmniej jeden bit, któregoś z argumentów ustawiony jest na 1. Możemy zatem powiedzieć, że wynikiem jest 1 gdy bit a **lub** bit b jest ustawiony na 1. Tabela wyników dla tego działania jest następująca:

a		
OR	b=0	b=1
b		
a=0	0	1
a=1	1	1

- *Przykład:*
0101 and 1100 = 1101

- **Operacja bitowa "albo"** (*ang. bitwise xor*)

Jest to operacja dwuargumentowa, zwana też alternatywą wykluczającą. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

a xor b

a ^ b

Wynikowy bit jest ustawiany na 1 tylko wówczas gdy dokładnie jeden bit, któregoś z argumentów ustawiony jest na 1. Możemy zatem powiedzieć, że wynikiem jest 1 gdy bit a **albo** bit b jest ustawiony na 1. Tabela wyników dla tego działania jest następująca:

a		
XOR	b=0	b=1
b		
a=0	0	1
a=1	1	0

- *Przykład:*
0101 and 1100 = 1001

- **Zaprzeczenie bitowe, dopełnienie bitowe** (*ang. bitwise not, complement*)

Jest to operacja jednoargumentowa. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

not a

~ a

Wynikowy bit jest ustawiany na 1 tylko wówczas gdy bit argumentu jest ustawiony na 0. Możemy zatem powiedzieć, że wynikiem jest 1 gdy bit a **nie jest** ustawiony na 1. Tabela wyników dla tego działania jest następująca:

a	NOT a
0	1
1	0

- *Przykład:*
not 0101 = 1010

- **Przesunięcie bitowe w lewo** (*ang. shift left*)

Jest to operacja dwuargumentowa. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

a shl b

a << b

Operacja polega na przesunięciu *a* o *b* bitów w lewo. Przy czym bity pojawiające się z prawej strony (uzupełniające przesunięcie) są ustawiane na 0. Operacja ta jest równoważna mnożeniu przez 2. Przesunięcie o 1 bit to przemnożenie *a* przez 2, przesunięcie o 2 bity to dwukrotne pomnożenie *a* przez 2, itd.

Przykład:
0101 shl 3 = 0101000

- **Przesunięcie bitowe w prawo** (*ang. shift right*)

Jest to operacja dwuargumentowa. W językach programowania bądź w pseudokodzie zapisywana jest z reguły jako:

a shr b

a >> b

Operacja polega na przesunięciu *a* o *b* bitów w prawo. Operacja ta jest równoważna dzieleniu całkowitemu przez 2. Przesunięcie o 1 bit to podzielenie *a* przez 2, przesunięcie o 2 bity to dwukrotne podzielenie *a* przez 2, itd.

Przykład:

010110 shr 3 = 010

Przykłady zastosowania:

- **Operacje na pojedynczym bicie**

Załóżmy, że mamy dany *rejestr*, w którym poszczególne bity odpowiadają za włączanie i wyłączanie poszczególnych funkcjonalności. Chcemy w prosty sposób operować na pojedynczym bicie nie zmieniając stanu pozostałych. W tym celu zdefiniujemy sobie *bit*, który będzie reprezentował bit, na którym dokonujemy operacji. Wszystkie jego bity ustawione są na zero, z wyjątkiem bitu na którym będziemy operować. Operacje zdefiniujemy następująco:

- o włączenie bitu: $rejestr = rejestr \text{ or } bit$
- o wyłączenie bitu: $rejestr = rejestr \text{ and } not(bit)$
- o przełączenie bitu w stan przeciwny: $rejestr = rejestr \text{ xor } bit$

wyłączenie → bitu oznacz, że gdy było 1 to ten bit staje się 0 a gdy było 0 to pozostaje 0

włączenie → bitu oznacz, że gdy było 0 to ten bit staje się 1, a gdy było 1 to pozostaje 1

przełączenie oznacza → gdy bit był 1 to staje się 0 a gdy był 0 to staje się 1

Przykład

Niech *bit* = 0100

Włączenie bitu dla *rejestr* = 1001: $1001 \text{ or } 0100 = 1101$

Wyłączenie bitu dla *rejestr* = 1101: $1101 \text{ and } not(0100) = 1101 \text{ and } 1011 = 1001$

Przełączenie bitu dla *rejestr* = 1001: $1001 \text{ xor } 0100 = 1101$

Przełączenie bitu dla *rejestr* = 1101: $1101 \text{ xor } 0100 = 1001$

- **Maskowanie**

Jest to operacja wybierania z większej liczby bitów tylko, tych, które faktycznie nas interesują. W tym przypadku posłużymy się operacją **i**. *Maska* będzie wybierać nam, które bity przy porównaniu są dla nas ważne (ustawione w masce na 1). Wówczas jeżeli chcemy porównać dwie dane: *dane_1* oraz *dane_2* użyjemy następującej konstrukcji:

if *dane_1* and *maska* = *dane_2* and *maska* **then...**

Doskonałym przykładem są tutaj sieci komputerowe i sposób ich adresacji. W protokole IP wyróżniamy część adresu mówiącą o sieci, w której znajduje się dany komputer, oraz część adresu mówiącą o numerze komputera w danej sieci. By zdecydować czy dany pakiet przesłać dalej czy znajduje się on w naszej sieci musimy sprawdzić czy część adresu, w której zawarty jest adres sieci jest zgodny z naszą siecią. Wykonuje się to za pomocą operacji **and** na adresie sieci oraz masce sieci. Zatem taka decyzja wygląda wówczas następująco:

if *adres_pakietu* and *maska_sieci* = *adres_moj* and *maska_sieci* **then...**

Przykład

Założmy, że:

$adres_pakietu = 11000000101010110000000100100101$

$adres_moj = 11000000101010110001000000010111$

$maska_sieci = 11111111111111110000000000000000$

Sprawdźmy czy dany pakiet adresowany jest do naszej sieci:

$adres_pakietu \text{ and } maska_sieci = adres_moj \text{ and } maska_sieci$

$11000000101010110000000100100101 \text{ and } 11111111111111110000000000000000$

$= 11000000101010110001000000010111 \text{ and }$

$11111111111111110000000000000000$

$11000000101010110000000000000000 = 11000000101010110000000000000000$

Równość jest spełniona zatem ten pakiet adresowany jest do naszej sieci.

- **Upakowywanie danych**

Czasem zachodzi potrzeba użycia danych o niestandardowym rozmiarze. Na przykład chcemy zapisywać bardzo wiele razy liczbę, której wartość mieści się w zakresie 0-3. Do zapisania takiego zakresu wystarczą nam dwa bity. Najmniejsza jednostka jaką możemy zapisać na dyku to bajt, który ma 8 bitów. Dlatego też, 4 takie dwubitowe wartości możemy upakować jednym bajcie. Zapis takich upakowanych wartości możemy zapisać następująco:

- odczyt upakowanej danej:

$odpakowana = (upakowana \text{ and } maska) \text{ shr } start_bit$

- zapis upakowanej danej będzie przebiegał dwuetapowo, najpierw wyczyszczenie miejsca na daną, a następnie jej zapis:

$upakowana = upakowana \text{ and } not(maska)$

$upakowana = upakowana \text{ or } (odpakowana \text{ shl } start_bit)$

Maska wskazuje nam miejsce gdzie należy upakować dane, natomiast *start_bit* jest numerem bitu (licząc od 0 od prawej), pod którym zaczyna się dana.

Przykład

Założmy, że mam 4 wartości: 01 11 10 01 wpisać do jednego bajta.

Kolejne maski dla tych danych zdefiniujemy następująco: 00000011, 00001100, 00110000, 11000000.

Kolejne bity startowe dla tych danych zdefiniujemy następująco: 0, 2, 4, 6.

Zapiszmy zatem drugą wartość do zmiennej *bajt* = 01101010, miejsce, gdzie zostanie zapisana dana pogrubiono. A więc najpierw czyszczenie miejsca pod nasze dane:

$bajt = bajt \text{ and } not(maska) = 01101010 \text{ and } not(00001100) = 01101010 \text{ and } 11110011 = 01100010.$

Teraz zapiszemy dane:

$bajt = bajt \text{ or } (dane \text{ shl } start_bit) = 01100010 \text{ or } (11 \text{ shl } 2) = 01100010 \text{ or } 1100 = 01101110.$

Spróbujmy teraz odczytać te dane: $dane = (bajt \text{ and } maska) \text{ shr } start_bit) = (01101110 \text{ and } 00001100) \text{ shr } 2 = 00001100 \text{ shr } 2 = 11.$

Przypisania

Operacje przypisania są dwuargumentowe i powodują przypisanie argumentu prawostronnemu argumentowi lewostronnemu. Taką najprostszą operację już poznaliśmy,

odbywa się ona przy wykorzystaniu operatora = (równa się). Jeśli napiszemy `a = 5`, oznacza to, że zmiennej `a` chcemy przypisać wartość 5.

Oprócz tego operatora `=`, w Javie występuje szereg operatorów łączonych tzn. takich, w których przypisaniu towarzyszy dodatkowa operacja, arytmetyczna lub bitowa.

Przykładowo taki zapis `a += b` tłumaczymy jako:

`a = a + b`.

Przykład 11(nie wpisujemy)

```
class Main
{
    public static void main(String args[])
    {
        int a = 15;
        a += 10;
        System.out.println("Zmienna a ma wartość = " + a);
    }
}
```

Wykonanie kodu z przykładu spowoduje przypisanie zmiennej `a` wartości 15, następnie dodanie do niej wartości 10 oraz wyświetlenie wyniku (`a = 25`) na ekranie.

W Javie występuje cała gama tych operatorów.

`arg1 op=arg2` oznacza działanie `arg1 = arg1 op arg2`,

czyli `a+=b` oznacza `a = a + b`,

`a *= b` oznacza `a = a * b`

Tabela Operatory przypisania i ich znaczenie w Javie

argument1	operator	argument2	znaczenia
x	=	Y	X=y
x	+=	Y	x = x + y
x	-=	Y	x = x - y
x	*=	Y	x = x * y
x	/=	Y	x = x / y
x	%=	Y	x = x % y
x	<<=	Y	x = x << y
x	>>=	Y	x = x >> y
x	>>>=	Y	x = x >>> y
x	&=	Y	x = x & y
x	=	Y	x = x y
x	^=	Y	x = x ^ y

Porównywania

Operatory porównania służą do porównywania argumentów. Wynikiem ich działania jest wartość logiczna true lub false.

Operator	Symbol
=	wynikiem jest true, jeśli argumenty są sobie równe
!=	wynikiem jest true, jeśli argumenty są różne
>	wynikiem jest true, jeśli argument prawostronny jest większy od lewostronnego
<	wynikiem jest true, jeśli argument prawostronny jest większy lub równy lewostronnemu
>=	wynikiem jest true, jeśli argumenty prawostronny jest mniejszy od lewostronnego
<=	wynikiem jest true, jeśli argumenty prawostronny jest mniejszy lub równy lewostronnemu

Zadanie ??

Nie ma ale będzie → zadanie dotyczy np. $a+=b$.

- nazwa klasy nazwisko_ucznia_z14 np. Kowalski_p12,

	<p>W lewo</p> <p>Na najmłodszą pozycję dopisywany jest bit o wartości zero, natomiast najstarszy bit jest tracony.</p> <p>W prawo</p> <p>Na najstarszą pozycję dopisywany jest bit o wartości zero, natomiast najmłodszy bit jest tracony.</p>
--	--

Przykład 10a

Temat: Operacje na bitach. Operatory AND OR XOR

Wykonaj:

- 1)Przepisz temat do zeszytu.
- 2)Uruchom przykład. Zmień nazwę klasy na nazwisko_P10a np.Kowalski_P10a. Podczas uruchamiania program pamiętaj że 0 i 1 muszą się wyświetlać pod operatorem w tym przypadku po znakiem „&”
- 3)Uzupełnij program o

Operacja bitowa "lub" (ang. *bitwise or*)

Operacja bitowa "albo" (ang. *bitwise xor*)

Wykonaj w formie jednej tabeli. Nie zwiększaj ilości wierszy w programie, dopisz dalsze części istniejących wierszy. Podczas uruchamiania program pamiętaj, że 0 i 1 muszą się wyświetlać pod operatorem AND OR XOR

- 4)Przepisz tabele prawdy z ekranu okienka CMD (gdy działa program, nie z instrukcji) do zeszytu, łącznie z tytułem, co przedstawia tabela.
- 5)Przepisz program do zeszytu.

```
public class przyklad_10a
{
    public static void main(String[] args)
    {

        int a,b;

        System.out.println("Operacja bitowa i");
        System.out.println(" a  b  a&b");
        a=1;b=1;
        System.out.println(" "+a+" "+b+" "+(a&b));
        a=0;b=1;
        System.out.println(" "+a+" "+b+" "+(a&b));
        a=1;b=0;
        System.out.println(" "+a+" "+b+" "+(a&b));
        a=0;b=0;
        System.out.println(" "+a+" "+b+" "+(a&b));
    }
}
```

Przykład 10b

Temat: Operacje na bitach. Demonstracja przesunięcia bitowego.

Wykonaj:

- 1)Przepisz temat do zeszytu.
- 2)Zapisz w zeszycie liczbę 90+numer w dzienniku w postaci dwójkowej w postaci np. $(245)_{10} = (11110101)_2$ Jeśli jest to konieczne dopisz zera na początku liczby tak aby było osiem bitów.
- 3)Dokonaj przesunięcia w tej liczbie o jeden bit w prawo i zapisz w zeszycie następująco:
Liczba w postaci binarnej przesunięta jeden bit w prawo → $(???????)_2$ Jeśli jest to konieczne dopisz zera na początku liczby tak aby było osiem bitów.
- 4)Dokonaj przesunięcia w tej liczbie (liczby 90+numer w dzienniku) o dwa bity w lewo i zapisz w zeszycie następująco:
Liczba w postaci binarnej przesunięta dwa bity w lewo → $(???????)_2$ Jeśli jest to konieczne dopisz zera na początku liczby tak aby było osiem bitów.
- 5)Uruchom przykład. Zmień nazwę klasy na nazwisko_P10b np.Kowalski_P10b (niektóre linie programu są podzielone po kopiowaniu będziesz musiał je połączyć poprzez skasowanie niewidocznego końca linii/akapitu)
- 6)Przepisz przykład i podkreśl w zeszycie innym kolorem linie programu, które przesuwają bity.
- 7)Wyciągnij wniosek zapisując w zeszycie:
Przesunięcie jeden bit w lewo to→
Przesunięcie dwa bity w prawo to→


```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import static java.lang.Math.*;

public class Zadanie_10b
{
    public static void main(String[] args) throws NumberFormatException, IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        int liczba1;
        System.out.print("podaj liczbę naturalną z przedziału <110;122>=");
        liczba1 = Integer.parseInt(reader.readLine());
        System.out.println("wczytana liczba");
        System.out.println("liczba1=" + ("+"+liczba1+"")10=("+Integer.toBinaryString(liczba1)+"2"));
        // przesuniecie o jeden bit lewo
        liczba1=liczba1<<1;
        System.out.println("liczba z przesuniętym jednym bitem w lewo czyli mnożenie przez ??");
        System.out.println("liczba1=" + ("+"+liczba1+"")10=("+Integer.toBinaryString(liczba1)+"2"));
        // przesuniecie o dwa bity prawo
        liczba1=liczba1>>2;
        System.out.println("liczba z przesuniętymi o dwa bity w prawo czyli dzielenie przez ??");
        System.out.println("liczba1=" + ("+"+liczba1+"")10=("+Integer.toBinaryString(liczba1)+"2"));

    }
}

```

Przykład 10c

Temat: Operacje na bitach. Uzupełnienie zerami do jednego bajta.

Wykonaj:

1)Przepisz temat do zeszytu.

2)Uruchom przykład. Zmień nazwę klasy na nazwisko_P10c np.Kowalski_P10c (niektóre linie programu są podzielone po kopiowaniu będziesz musiał je połączyć poprzez skasowanie niewidocznego końca linii/akapitu)

3)Przeczytaj algorytm działania programu:

Przy wczytaniu liczby z przedziału <0;255> i zamianie na liczbę binarną. Liczba binarna może nie zajmować 1 bajta, czyli ośmiu znaków. Należy wtedy uzupełnić ją do ośmiu znaków (ośmiu jedynek lub zer).

-obliczamy jak długa jest liczba po zamianie na liczbę binarną, czyli ile ma znaków

-stosujemy pętlę dodającą do siebie zera (traktowane jako znaki) ilość zer to różnica między osiem a długością liczby binarnej,

-zera dodajemy do liczby w postaci binarnej.

4)Zapisz znaczenie następujących instrukcji:

- Integer.toBinaryString(liczba1)
- length
- concat

5)Przepisz program do zeszytu.

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import static java.lang.Math.*;
public class przyklad_10c
{
    public static void main(String[] args) throws NumberFormatException, IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        int liczba1;
        String liczba1_bin, zera;
        System.out.print("podaj liczbę <0;255> liczbę =");
        liczba1 = Integer.parseInt(reader.readLine());
        System.out.println("liczba1=" + ("+"+liczba1+"")10 = ("+Integer.toBinaryString(liczba1)+")2");
        liczba1_bin=Integer.toBinaryString(liczba1);
        zera="";
        for (int i=1; i<9-Integer.toBinaryString(liczba1).length();i++)
        {
            zera=zera.concat("0");
        }
        zera=zera.concat(liczba1_bin);
        System.out.println("liczba z uzupełnionym zerami do jednego bajta");
        System.out.println("liczba1="+("+"+zera+"")2");
    }
}

```

Koniec przykładu.

Określanie podsieci

Podsieć jest wydzielana częścią z sieci LAN. Jest to logiczne (nie fizyczne) oddzielenie komputerów. W przypadku wydzielenia podsieci komputery nie będą mogły w prosty sposób komunikować się między sobą.

Określanie adresu podsieci

Dzięki stosowaniu maski podsieci można podzielić sieć na kilka podsieci.

Dla adresu IP komputera 195.25.22.210 i maski podsieci 255.255.255.224 ustal adres podsieci.

Rozwiązanie

1) zapisujemy maskę podsieci w postaci binarnej

11111111 11111111 11111111 11100000 → 255.255.255.224

Oznacza to, że trzy pierwsze bity czwartego oktetu reprezentują pole podsieci.

2) zapisujemy adres IP w postaci binarnej

11000011 00011001 00010110 11010010 → 195.25.22.210

3) Dla określenia adresu podsieci trzeba zestawić binarnie adres IP i maskę w procesie AND (jest to **iloczyn logiczny**, który charakteryzują następujące właściwości: jeżeli zestawiamy dwie jedyńki, wynikiem jest 1; jeżeli zestawiamy 0 i 1, wynikiem jest 0; jeżeli zestawiamy dwa zera, wynikiem jest 0.)

```
      11000011 00011001 00010110 11010010
      11111111 11111111 11111111 11100000
-----
AND    11000011 00011001 00010110 11000000
```

adres podsieci jest 195.25.22.192

Adres rozgłoszeniowy

Adresy rozgłoszeniowe są wykorzystywane do wysyłania informacji przez komputery w sieci(podsieci) do wszystkich urządzeń danej sieci (podsieci).

Obliczanie adresu rozgłoszeniowego dla określonej sieci(podsieci).

Dla adresu IP komputera 195.25.22.210 i maski podsieci 255.255.255.224 ustal adres rozgłoszeniowy.

Rozwiązanie

1) Oblicz adres podsieci dla danego adresu dowolnego komputera pracującego w tej sieci.
11000011 00011001 00010110 11000000 → 195.25.22.192

2) Zaneguj adres maski sieci. Negowanie polega na zastąpieniu zer jedynekami a jedyńki zerami w całej masce podsieci zapisanej binarnie.

11111111 11111111 11111111 11100000 → 255.255.255.224 maska podsieci

00000000 00000000 00000000 00011111 →zanegowana maska podsieci

3) Dodaj binarnie adres sieci danego komputera do zanegowanej maski podsieci.

Dodawanie binarne (1+1=0 przeniesienie 1 1+0=1 0+1=1 0+0=0)

11000011 00011001 00010110 11000000 → 195.25.22.192

00000000 00000000 00000000 00011111 →zanegowana maska podsieci

11000011 00011001 00010110 11011111 →195.25.22.223 adres rozgłoszeniowy

Zadanie 14a1

Temat: Wyznaczanie adresu sieci. Wyznaczanie adresu rozgłoszeniowego.

Wykonaj:

1)Zapisz w zeszycie co oznaczają następujące pojęcia:

-co to jest podsieć

-co to jest maska podsieci

2)Przeczytaj w jaki sposób wyznaczamy:

-adres podsieci

3)Napisz program wyznaczający adres podsieci:

a)Wczytaj adres komputera jako cztery liczby z zakresu <0,255> → podczas wczytywania użyj klasy adresowej C.

b)Wypisz wczytany adres komputera w postaci np. 195.173.10.123

c)Wczytaj maskę podsieci jako cztery liczby z zakresu <0,255> → wpisz możliwe maski podsieci

d)zamień osiem zmiennych (cztery dla adresu komputera i cztery dla maski podsieci) uzupełnij zerami tak aby tworzyły 1 bajt (jak to zrobić patrz przykład poprzedni).

e) oblicz w czterech bajtach adres podsieci na podstawie wartości adresu komputera i maski podsieci. Uzupełnij zerami cztery zmienne adresu posieci do jednego bajta. Pamiętaj o zerowaniu zmiennej *zero*. Czyli o przypisaniu tej zmiennej pustego ciągu znaków "".

f) wyprowadź na ekran

11000011 00011001 00010110 11010010 ← adres komputera

11111111 11111111 11111111 11100000 ← maska posieci

AND 11000011 00011001 00010110 11000000

g) wyświetl otrzymany adres podsieci w postaci np. 195.25.22.192

h) zapisz w zeszycie tylko te linie programu, które wyświetlają:

- wczytany adres komputera w postaci np. 195.173.10.123
- realizacja AND dla jednego bajta
- wyświetlenie adresu komputera w postaci binarnej.

Zadanie 14a2 (Nieobowiązkowe → gdy chcesz dostać ocenę celującą z tabelki)

Temat: Wyznaczanie adresu rozgłoszeniowego.

Wykonaj:

1) Zapisz w zeszycie co oznaczają następujące pojęcia:

-co to jest adres rozgłoszeniowy

2) Przeczytaj w jaki sposób wyznaczamy:

-adres rozgłoszeniowy

3) Napisz program wyznaczający adres rozgłoszeniowy:

-Wczytaj adres komputera jako cztery liczby z zakresu <0,255> → podczas wczytywania użyj klasy adresowej C.

-Wypisz wczytany adres komputera w postaci np. 195.173.10.123

-Wczytaj maskę podsieci jako cztery liczby z zakresu <0,255> → wpisuj możliwe maski podsieci

-zamień osiem zmiennych (cztery dla adresu komputera i cztery dla maski podsieci) uzupełnij zerami tak aby tworzyły 1 bajt (jak to zrobić patrz przykład poprzedni).

-Uzupełnij zerami cztery zmienne adresu posieci do jednego bajta. Pamiętaj o zerowaniu zmiennej *zero*. Czyli o przypisaniu tej zmiennej pustego ciągu znaków "".

-oblicz w czterech bajtach adres rozgłoszeniowy na podstawie wartości adresu komputera i maski podsieci. Uzupełnij zerami cztery zmienne adresu rozgłoszeniowego do jednego bajta.

-wyprowadź na ekran

11000011 00011001 00010110 11000000 ← adres komputera

11111111 11111111 11111111 11100000 ← maska posieci

00000000 00000000 00000000 00011111 ← zanegowana maska podsieci

SUMA 11000011 00011001 00010110 11011111 ← adres rozgłoszeniowy

-wyświetl adres rozgłoszeniowy w postaci np. 195.25.22.223

Zadanie 14a

Temat: Obliczanie ilości punktów kratowych z użyciem pętli podwójnej.

Punkt kratowy to punkt, którego współrzędne w układzie kartezjańskim są liczbami całkowitymi.

Przykłady punktów kratowych:

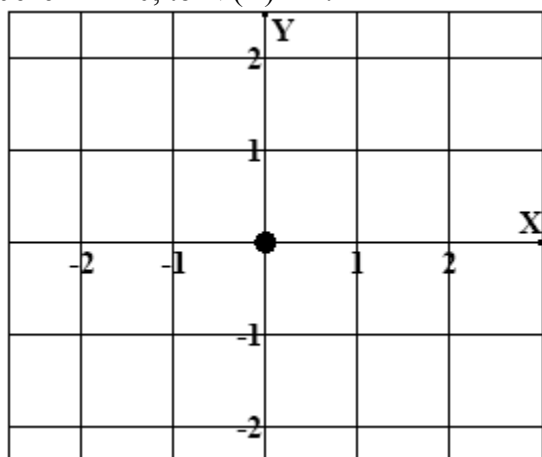
$(-100, 101)$, $(1, 1)$, $(0, 0)$, $(-1, -3)$.

Rozważamy koła o środku w początku układu współrzędnych. Dla nieujemnej liczby rzeczywistej R przez $K(R)$ oznaczmy koło o promieniu R (brzeg koła należy do koła).

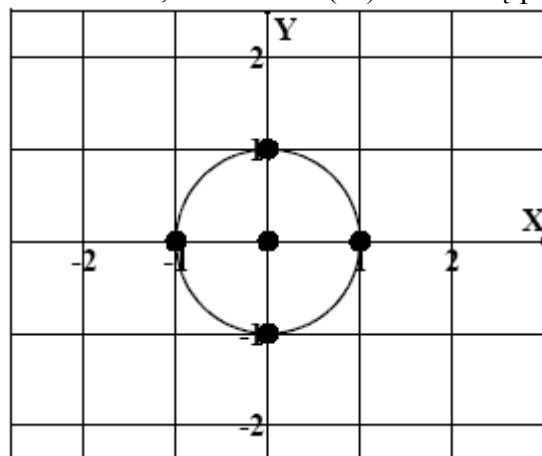
Niech $N(R)$ będzie liczbą punktów kratowych zawartych w kole $K(R)$.

Przykłady:

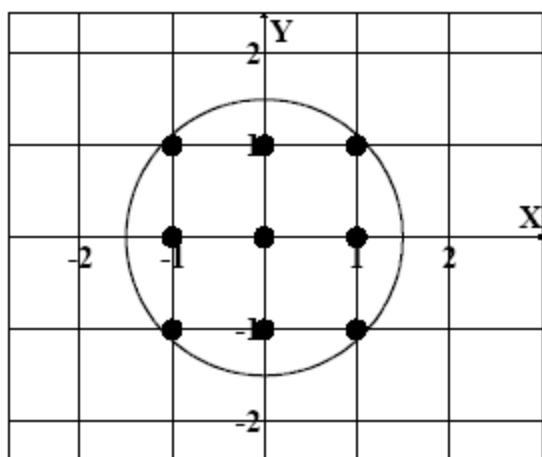
Jeżeli $R = 0$, to $N(R) = 1$.



Jeżeli $R = 1$, to w kole $K(R)$ mieści się pięć punktów kratowych, czyli $N(R) = 5$.



Jeżeli $R = 1,5$, to w kole $K(R)$ mieści się dziewięć punktów kratowych, zatem $N(R) = 9$.



Do wykonania

a) Zapisz specyfikację w zeszycie.

Specyfikacja:

Dane: R – promień koła o środku znajdującym się w początku układu współrzędnych $(0,0)$; liczba całkowita nieujemna.

Wynik: liczba całkowita $N(R)$ – liczba punktów kratowych zawierających się w kole o środku $(0,0)$ i promieniu R

b) Napisz program obliczający liczbę punktów kratowych zawierających się w kole o promieniu R . Użyj zmiennych sterujących i i j – trzy pierwsze litery nazwiska ucznia oraz $j_trzy_pierwsze_litery_nazwiska_ucznia$ np. i_kow oraz j_kow .

c) Uzupełnij i zapisz w zeszycie poniższą tabelę używając napisany program:

Promień koła R	Liczba punktów kratowych $N(R)$
2,01	
4,50	
100+numer_w_dzienniku	
1000+numer_w_dzienniku	

Rozwiązanie:

Należy przeszukać obszar w postaci kwadratu w układzie XOY $(-1-R) \leq x \leq (1+R)$ $(-1-R) \leq y \leq (1+R)$. W tym celu należy stworzyć dwie pętle (jedna w drugiej, czyli pętla podwójna). Pętla podwójna będzie generować współrzędne wszystkich punktów o zmiennych całkowitych wewnątrz tego kwadratu. Należy zwrócić uwagę, że zmienna R jest liczbą rzeczywistą a nie całkowitą dlatego przed użyciem w pętli należy zamienić ją na całkowitą. np. $(\text{int})(-R_kow-1)$. Dla wszystkich wygenerowanych punktów sprawdź czy odległość wygenerowanego punktu od początku układu współrzędnych (punkt $(0,0)$) jest mniejsza lub równa od R . Jeśli tak to zwiększ licznik (na początku zadania musi być zerowany). Po wykonaniu zadania wyświetl licznik.

Jeśli wygenerowany punkt ma współrzędne (i_kow, j_kow) to warunek ma postać:

$$\sqrt{(i_kow)^2 + (j_kow)^2} \leq R_kow$$

Proste tablice w Javie

Uwagi:

- 1) Tablice w Javie są obiektami. Aby móc skorzystać z tablicy, musimy po pierwsze zadeklarować zmienną tablicową, a następnie utworzyć samą tablicę.
- 2) Pierwszy element tablicy ma numer zero.

Sposoby deklaracji tablic:

- 1) Możemy jednocześnie zadeklarować i utworzyć tablicę, korzystając z konstrukcji:

```
typ_tablicy nazwa_tablicy[] = new typ_tablicy[liczba_elementów];
```

- 2) Rozbicie deklaracji tablicy na dwie instrukcje:

```
typ_tablicy nazwa_tablicy[];  
/*tutaj mogą znaleźć się inne instrukcje*/  
nazwa_tablicy = new typ_tablicy[liczba_elementów];
```

Jak widzimy, pomiędzy deklaracją o utworzeniu tablicy można umieścić również inne instrukcje. W przypadku prostych programów wykorzystuje się ten pierwszy sposób.

Przykład 12

- 1) Wpisz teorię:

- Czym są tablice w Java?
- Co należy wykonać, aby skorzystać z tablicy w Java?
- Jaki numer ma pierwszy element tablicy?
- Zapisz dwa sposoby deklaracji tablic.

- 2) Poniżej jest przykład. Na podstawie tego przykładu poniżej treści zadania:

- nazwa klasy nazwisko_ucznia_p12 np. Kowalski_p12,
- nazwie tab_cztery_pierwsze litery nazwiska ucznia np. tab_kowa,
- zarezerwuj tablicę jedno wymiarową o wymiarze liczba_liter_nazwiska,
- do pierwszego elementu tablicy wpisz wartość numer_w_dzienniku,
- do ostatniego elementu tablicy wpisz wartość liczba_liter_imienia,
- wypisz na ekranie zawartość ostatniego i pierwszego elementu tablicy.

Przykład poniższy musisz zmienić (nazwa klasy, wymiar tablicy i wartości tablicy)

```
class Main // zmien nazwe klasy  
{  
    public static void main (String args[])  
    {  
        int tab[] = new int[1];  
        tab[0] = 10;  
        System.out.println("Pierwszy element tablicy ma wartość: " + tab[0]);  
    }  
}
```

Przykład 13

Poniżej jest przykład. Na podstawie tego przykładu:
zarezerwuj tablicę:

- nazwa klasy nazwisko_ucznia_p13 np. Kowalski_p13,
- jednowymiarową,
- nazwie tab_trzy_pierwsze litery nazwiska ucznia,
- wymiar tablicy numer_z_dziennia+5,
- zawartość tablicy to numer_z_dziennia

```
class Tablice_w11 // zmien nazwe klasy
{
    public static void main (String args[])
    {
        short tablica[] = new short[12];
        for(short i = 0; i < 11; i++)
        {
            tablica[i] = 100;
        }
        for(short i = 0; i < 11; i++)
        {
            System.out.println("tablica[" + i + "] = " + tablica[i]);
        }
    }
}
```

Zadanie 15

1)Teoria do zadania.

Dzięki temu, że tablice są obiektami, każda z nich posiada właściwości length, która przechowuje rozmiar tablicy. Właściwość ta może być tylko odczytywana. Jeśli zastosujemy konstrukcję w postaci:

`nazwa_tablicy.length`

Otrzymamy rozmiar dowolnej tablicy. Należy pamiętać o numerowaniu poszczególnych komórek tablicy od zera, zatem jeśli chcemy odwołać się do ostatniego elementu tablicy, należy odwołać się do indeksu o wartości (length – 1).

2)Wykonaj:

Zapisz w zeszycie:

- czym jest lenght do odniesieniu do tablicy i co należy zrobić aby z tego skorzystać,
- jak odwołujemy się do ostaniej komórki tablicy.

3) nazwa klasy nazwisko_ucznia_z15 np. Kowalski_z15,

treść programu→zarezerwuj tablicę:

- jednowymiarową,
- nazwie tab_cztery_pierwsze litery nazwiska ucznia,
- wymiar tablicy numer_z_dziennia+10,
- użyj pętli do generowania zawartości tablicy oraz wyświetlania ich zawartości, użyj length
- zawartości tablicy to:

`tab[0]=0`

tab[1]=1
tab[2]=2
.....
czyli tab[i]=i

Zadanie 15a

Część teoretyczna zadania

Wykonaj:

- a) zapisz w zeszytcie co to jest algorytm, wymień sposoby przedstawiania algorytmów, cechy charakterystyczne poprawnego algorytmu.
- b) zapisz w zeszytcie co to sortowanie, wymień metody sortowania oraz na czym polega metoda sortowania bąbelkowego.
- c) przeczytaj ze zrozumieniem metodę sortowania bąbelkowego
(dla ciągu 1 2 9 7 10 8 4 przebiegi i krok → patrz poniżej)

Teoria w celu rozwiązania zadania.

Algorytm → jest to pewien ciąg czynności, który prowadzi do rozwiązania danego problemu.

Algorytmy można przedstawiać m.in. następującymi sposobami:

- słowny opis
- schemat blokowy
- lista kroków
- drzewo algorytmu
- drzewo wyrażeń
- w pseudojęzyk
- w język programowania.

Cechy charakterystyczne poprawnego algorytmu:

1. **Poprawność** - dla każdego przypisanego zestawu danych, po wykonaniu skończonej liczby czynności, algorytm prowadzi do poprawnych wyników.
2. **Jednoznaczność** - w każdym przypadku zastosowania algorytmu dla tych samych danych otrzymamy ten sam wynik.
3. **Szczegółowość** - wykonawca algorytmu musi rozumieć opisane czynności i potrafić je wykonywać.
4. **Uniwersalność** - algorytm ma służyć rozwiązywaniu pewnej grupy zadań, a nie tylko jednego zadania. Przykładowo algorytm na rozwiązywanie równań w postaci $ax + b = 0$ ma je rozwiązać dla dowolnych współczynników a i b , a nie tylko dla jednego konkretnego zadania, np. $2x + 6 = 0$

Sortowanie — oznacza proces porządkowania według pewnego klucza np.: od największego do najmniejszego, najmniejszego do największego, alfabetycznie itp.

Sortowanie może odbywać się według wielu różnych algorytmów, różniących się ilością wykonywanych operacji oraz szybkością działania.

Metody sortowania.

Sortowanie bąbelkowe (angielskie bubble sort).

Jest to sortowanie polegające na przeglądaniu po kolei elementów porządkowanego ciągu i zamienianiu miejscami sąsiadujących elementów tak, aby spełniały relację porządkującą; w ten sposób elementy mniejsze ("lżejsze") przesuwają się na początek ciągu. Dla n elementów ciągu złożoność sortowania bąbelkowego wynosi $O(n^2)$. Sortowanie bąbelkowe jest sortowaniem stabilnym.

Sortowanie bąbelkowe z kresem górnym

Może się zdarzyć sytuacja, gdy tablica będzie już uporządkowana, zanim zrobimy $n-1$ przebiegów lub po k przebiegach więcej niż k liczb od końca będzie na swoim miejscu. Ten problem także da się ominąć. Wystarczy do naszego programu wprowadzić zmienną, która będzie reprezentować w której pozycji nastąpiła ostatnia zamiana elementów tablicy. Ta zmienna to jakby kres następnego przebiegu. Ten wariant sortowania nazywa się sortowaniem bąbelkowym z kresem górnym.

Sortowanie przez wybór (angielskie selectsort)

Jest to sortowanie, w którym w każdym kroku algorytmu znajduje się najmniejszy element w sortowanym ciągu, po czym przenosi się ten element na kolejną pozycję do ciągu wynikowego (przez zamianę elementów miejscami).

Sortowanie przez wstawianie (insertion sort)

Sortowanie przez wstawianie to algorytm, którego czas działania wynosi $O(n^2)$. Jest on skuteczny dla małej ilości danych. Jest to jeden z prostszych i jeden z bardziej znanych algorytmów sortowania. Jest on stabilny i nie wymaga dodatkowej pamięci (działa w miejscu). Często stosujemy go podczas gry w karty, biorąc je ze stołu. Biorąc po jednej karcie ze stołu wstawiamy ją w odpowiednie miejsce do kart, które mamy w ręce.

Sortowanie przez zliczanie (counting sort)

Sortowanie przez zliczanie jest jednym z najszybszych algorytmów sortowania danych, a przy tym bardzo prostym do wytłumaczenia. Algorytm ten działa w czasie $O(n)$, tak więc jest to sortowanie w czasie liniowym. Mimo swoich zalet, sortowanie przez zliczanie ma swoje dwie poważne wady. Po pierwsze - do tego sortowania potrzebna jest dodatkowa pamięć (czyli nie jest to sortowanie w miejscu), a po drugie - tym sposobem można sortować tylko liczby całkowite z określonego przedziału. Niewiadomo dlaczego, ale algorytm ten mimo swojej prostoty nie jest powszechnie znany, a tym bardziej używany.

Sortowanie pozycyjne (radix sort)

Stosowane jest do sortowania elementów, które składają się z szeregu pozycji (mogą to być liczby, gdzie pozycjami są poszczególne cyfry; wyrazy - w tym przypadku są to poszczególne litery; mogą to także być inne dane, np. daty). Algorytm ten wymaga użycia innego algorytmu sortowania podczas swego działania, co ważne sortowanie to musi być stabilne. Gdy jako dodatkowego algorytmu sortowania użyjemy sortowania przez zliczanie to algorytm sortowania pozycyjnego działa w czasie $O(n)$. Sortowanie pozycyjne stosuje się nie tylko do sortowania liczb, czy wyrazów. W ten sposób możemy także sortować np. daty. Musimy jednak pamiętać, aby sortować od pozycji najmniej znaczących. W przypadku dat - najpierw sortujemy je według dni, potem według miesięcy, a na końcu według lat. Sortowanie pozycyjne możemy także zastosować do sortowania rekordów baz danych. Na przykład chcemy posortować książkę telefoniczną według nazwisk, a w razie gdyby się one powtarzały to według imion, a w przypadku identyczności imion i nazwisk według numeru telefonu. Aby otrzymać taki wynik powinniśmy tę książkę telefoniczną posortować najpierw według numeru telefonu, potem według imion, a na końcu według nazwisk. Złożoność obliczeniowa takiego sortowania pozycyjnego na pewno nie będzie $O(n)$. Wynika to z tego, że do posortowania np. nazwisk trudno jest użyć sortowania przez zliczanie.

Sortowanie wyrazów

Z sortowaniem wyrazów jest podobnie jak z sortowaniem liczb. W tym przypadku pozycjami są poszczególne litery danego wyrazu. Jednakże jest pewna różnica jeśli chodzi o sortowanie wyrazów różnej długości. W tym przypadku odpowiednią ilość znaków dopisujemy za wyrazem, a nie jak to było w przypadku liczb - przed. Znak ten musi być traktowany jako wyżej w alfabecie, niż wszystkie inne - może to być na przykład spacja.

Sortowanie QuickSort

Jest to jeden z najpopularniejszych algorytmów sortowania. Wpłynęły na to dwie rzeczy. Po pierwsze jest ono bardzo szybkie (jak sama nazwa wskazuje), a po drugie - proste do wytłumaczenia i implementacji. Pesymistyczny czas jego działania wynosi $O(n^2)$, a średni $O(n \cdot \lg(n))$. Mimo tego w praktyce jest to najszybszy algorytm sortowania dużych tablic danych.

Sortowanie szybkie opiera się na technice "dziel i zwyciężaj". Wejściowa tablica jest dzielona (po przestawieniu niektórych z jej elementów) na dwie mniejsze. Każdy element pierwszej tablicy nie jest większy niż każdy element drugiej tablicy. Liczbę, według której wykonuje się podziału to najczęściej pierwszy element tablicy. Następnie dla tych dwóch podtablic wywołany jest rekurencyjnie ten sam algorytm. Wywołania rekurencyjne kończą się aż któraś z kolejnych podtablic będzie zawierała tylko jeden element. QuickSort działa w miejscu.

Inna odmiana QuickSort

Wyżej wspomniane jest, że algorytm szybkiego sortowania ma pesymistyczny czas działania $O(n^2)$. Występuje to w przypadku, gdy tablica wejściowa jest posortowana odwrotnie, tzn. jej wyrazy stanowią ciąg nierosnący.

Opis do algorytmu przedstawionego poniżej.

Metoda **bąbelkowa** polega: (porządkowanie od najmniejszego do największego–lub odwrotnie), ustawiamy się na pierwszym wyrazie ciągu i porównujemy go z drugim jeśli pierwszy jest większy od drugiego to zamieniamy je miejscami, następnie umieszczamy się na drugim i porównujemy z trzecim dokonując przestawienia lub nie. Porównując parami dochodzimy do końca ciągu (ustawiamy się na przedostatnim wyrazie). Jeśli podczas przechodzenia nastąpiła zmiana to musimy to zapamiętać jeśli nie to znaczy, że ciąg jest ustawiony prawidłowo. Po przejściu całego ciągu ustawiamy się ponownie na początek i sprawdzamy czy w poprzednim przechodzeniu było przestawienie jeśli nie było zamian to kończymy program jeśli było przestawienie to proces powtarzamy tyle razy aż nie będzie przestawienia.

Tablicy uporządkować w porządku rosnącym.

9	2	7	10	8	4
---	---	---	----	---	---

przebieg porządkowania będzie następujący

pierwszy przebieg

zamienione elementy

krok 1 2 9 7 10 8 4

9 2

krok 2 2 7 9 10 8 4

9 7

krok 3 2 7 9 8 10 4

10 8

krok 4 2 7 9 8 4 10

10 4

drugi przebieg

zamienione elementy

krok 1 2 7 8 9 4 10

9 8

krok 2 2 7 8 4 9 10

9 4

trzeci przebieg

zamienione elementy

krok 1 2 7 4 8 9 10

8 4

czwarty przebieg

zamienione elementy

krok 1 2 4 7 8 9 10

7 4

w piątym przebiegu nie nastąpiła zmiana tzn., że można skończyć sortowanie.

Treść praktyczna zadania

Temat: Dokonaj sortowania tablicy jednowymiarowej metodą bąbelkową z użyciem listy kroków, która jest zapisana poniżej.

Rozwiązanie:

- 1) nazwa klasy nazwisko_ucnia_z15a np. Kowalski_z15a,
- 2) Przepisz do zeszytu listę kroków dla sortowania bąbelkowego.
- 3) Przepisz wzór do przestawiania elementów w ciągu (ten ze skrytką).
- 4) Utwórz tablicę d_trzy_pierwsze_litery_nazwiska np. d_kow o siedmiu elementach. Nadaj elementom tablicy następujące wartości:
d_kow[0]= numer_z_dziennika;
d_kow[1]= liczba_liter_imienia;
d_kow[2]= liczba_liter_nazwiska;
d_kow[3]= –miesiąc_urodzenia;
d_kow[4]= dzień_urodzenia;
d_kow[5]= numer_but
d_kow[6]= wag_w_kg
- 5) Wyświetl tablicę d_kow
- 6) Dokonaj sortowania wg listy kroków
Użyj zmiennej SKRYTKA_trzy_pierwsze_litery. Użyj dwóch pętli i_nazwisko_ucnia, j_nazwisko_ucnia.
- 7) Wyświetl tablicę po sortowaniu.

Algorytmy mogą być przedstawione z użyciem listy kroków. Poniższa lista kroków przedstawia sortowanie metodą bąbelkową.

KROK1: Dla $j = 1, 2, \dots, n-1$: wykonaj KROK2
 KROK2: Dla $i = 1, 2, \dots, n-1$: jeśli $d[i] > d[i+1]$, to $d[i] \leftarrow d[i+1]$
 KROK3: Zakończ algorytm.

Opis:

Zapis oznacza $d[i] \leftarrow d[i+1]$, że należy zamienić miejscami elementy $d[i]$ i $d[i+1]$. Zamianę można uzyskać poprzez:

SKRYTKA = $d[i]$;

$d[i] = d[i+1]$;

$d[i+1] = \text{SKRYTKA}$;

gdzie SKRYTKA jest zmienną pomocniczą a $d[i]$ jest aktualnym elementem.

Pamiętaj, że pierwszy element tablicy to zero np. $d[0]$. Musisz uwzględnić ten fakt w konstrukcji pętli tak, aby zacząć od elementu zero czyli wartość początkowa zmiennej sterującej pętlą ma wartość zero. Musisz pomyśleć również jaka będzie ostatnia wartość pętli sterującej (w liście kroków jest to $n-1$).

Koniec zadania 15a

Tablice dwuwymiarowe

Deklaracja tablicy o stałych wartościach

`typ_tablicy nazwa_tablicy[][] = { {wartość1, wartość2, ..., wartośćn},
 { wartość1, wartość2, ..., wartośćn} }`

Zapis ten można rozbić na kilka linii, w celu zwiększenia jego czytelności:

```
typ_tablicy nazwa_tablicy[][] = {
{wartość1, wartość2, ..., wartośćn},
{ wartość1, wartość2, ..., wartośćn}
}
```

Przykład 14

Wykonaj:

1) Zapisz w zeszycie:

- Deklaracja tablicy o stałych wartościach
- Deklaracja tablicy dwuwymiarowej jako obiektu

2) Wykonaj praktycznie

- nazwa klasy nazwisko_ucznia_p14 np. Kowalski_p14,
- treść programu → Utwórz tablicę dwa wiersze i cztery komórki i wpisz początkowe wartości do tablicy (narysuj tabelę z wypełnionymi danymi w zeszycie)

1+nr_z_dziennika	2+nr_z_dziennika	3+nr_z_dziennika	4+nr_z_dziennika
5+nr_z_dziennika	6+nr_z_dziennika	7+nr_z_dziennika	8+nr_z_dziennika

- nazwie tab_cztery_pierwsze litery nazwiska ucznia,

Następnie wypisz dane z tabeli wyraz po wyrazie. Wzoruj się na przykładzie poniżej.

```

class Main // zmien nazwe klasy
{
    public static void main (String args[])
    {
        int tab[][] = {
            {1, 2, 3, 4},
            {5, 6, 7, 8}
        };
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 4; j++){
                System.out.println("tab[" + i + "][" + j + "] = " + tab[i][j]);
            }
        }
    }
}

```

Deklaracja tablicy dwuwymiarowej jako obiektu

Możemy jednocześnie zadeklarować i utworzyć tablicę, korzystając z konstrukcji:

```
typ_tablicy nazwa_tablicy[][] = new typ_tablicy[liczba_elementów] [liczba_elementów];
```

Bądź też mógł rozbić tę czynność na dwie instrukcje:

```

typ_tablicy nazwa_tablicy[][];
/*tutaj mogą znaleźć się inne instrukcje*/
nazwa_tablicy = new typ_tablicy[liczba_elementów] [liczba_elementów];

```

Tablice taką możemy wypełnić danymi przy użyciu zagnieżdżonych pętli for, tak jak jest zaprezentowane na przykładzie.

Przykład 15

1)Wykonaj praktycznie

- nazwa klasy nazwisko_ucznia_p15 np. Kowalski_p15,
- Zapisz w tablicy dwuwymiarowej o wymiarach:

ilości wierszy → [reszta z dzielenia (numer_miesiąca_urodzenia) przez trzy]+2

ilość kolumn → [reszta z dzielenia (numer_dnia_urodzenia) przez cztery]+2

wierszami wartości będąc kolejnymi liczbami naturalnymi poczynając od wartości numer_w_dzienniku+20

Wypisz elementy tablicy wyraz po wyrazie następnie wierszami.

Wzoruj się na przykładzie poniżej.

```

class Tab_dwu_p2 // zmien nazwe klasy
{
    public static void main (String args[])
    {
        int tab[][] = new int[2][4];
        int licznik = 1;
        for(int i = 0; i < 2; i++)
        {
            for(int j = 0; j < 4; j++)

```

```

        {
            tab[i][j] = licznik++;
        }
    }
System.out.println("wypisanie tablicy wyraz po wyrazie");
for(int i = 0; i < 2; i++)
{
    for(int j = 0; j < 4; j++)
    {
        System.out.println("tab[" + i + "][" + j + "] = " + tab[i][j]);
    }
}
System.out.println("wypisanie tablicy wierszami" );
for(int i = 0; i < 2; i++)
{
    System.out.println(tab[i][0]+" "+tab[i][1]+" "+tab[i][2]+" "+tab[i][3]);
}
}
}

```

ZADANIE 16

1)Wykonaj praktycznie

- nazwa klasy nazwisko_ucznia_z16 np. Kowalski_z16,

Napisać program generowania tablicy 4x6 gdzie wartość wyrazu jest równa sumie indeksów tego wyrazu czyli :

$$a_4_lityry_nazwiska[i][j]=i+j$$

czyli np. $a_4_lityry_nazwiska[2][3]=2+3=5$

Wykorzystaj poprzedni przykład modyfikując (zapisując inaczej) linię kodu w przykładzie poprzednim:

```
tab[i][j] = licznik++;
```

Wydrukuj tablicę na dwa sposoby:

- wyraz po wyrazie
- wierszami.

Uwaga: Przy rozwiązywaniu następnego zadania wykorzystaj treść tego przykładu.

ZADANIE 17

1)Wykonaj praktycznie

- nazwa klasy nazwisko_ucznia_z17 np. Kowalski_z17,

Napisać program generowania tablicy 7*7 która będzie wyglądać następująco :

```

1 0 0 0 0 0 0      gdy i >= j to a[i,j]=1
1 1 0 0 0 0 0      i < j to a[i,j]=0
1 1 1 0 0 0 0
1 1 1 1 0 0 0
1 1 1 1 1 0 0
1 1 1 1 1 1 0
1 1 1 1 1 1 1

```

Wydrukuj tablicę dwoma sposobami wyraz po wyrazie oraz wierszami .

ZADANIE 18

1) Wykonaj praktycznie

Rozwiązać układ dwóch równań liniowych w postaci. Do rozwiązania użyj tablic i pętli.
np.

$$A[0][0] * X + A[0][1] * Y = A[0][2]$$

$$A[1][0] * X + A[1][1] * Y = A[1][2]$$

$$\begin{cases} 2 * x + 3 * y = 5 \\ -4 * x + 8 * y = 0 \end{cases}$$

i tak

$$\begin{array}{lll} A[0][0]=2 & A[0][1]=3 & A[0][2]=5 \\ A[1][0]=-4 & A[1][1]=8 & A[1][2]=0 \end{array}$$

Tak więc w celu rozwiązania układu równań konieczne będzie użycie tablicy o dwóch wierszach i trzech kolumnach czyli konieczna będzie rezerwacja tablicy A[1][2].

W przypadku tablic dwuwymiarowych pierwsza liczba jest ilością wierszy, a druga ilością kolumn.

Przydatna teoria.

Wyznaczniki obliczamy stosując wzory:

$$W = \begin{vmatrix} A[0][0] & A[0][1] \\ A[1][0] & A[1][1] \end{vmatrix} = A[0][0] * A[1][1] - A[0][1] * A[1][0]$$

$$W_x = \begin{vmatrix} A[0][2] & A[0][1] \\ A[1][2] & A[1][1] \end{vmatrix} = A[0][2] * A[1][1] - A[1][2] * A[0][1]$$

$$W_y = \begin{vmatrix} A[0][0] & A[0][2] \\ A[1][0] & A[1][2] \end{vmatrix} = A[0][0] * A[1][2] - A[1][0] * A[0][2]$$

Rozwiązywalność układu równań (możliwe przypadki A) lub B) lub C):

A) gdy ($W \neq 0$) układ ma jedno rozwiązanie

gdzie: $X = W_x / W$ $Y = W_y / W$

W —wyznacznik główny układu równań

W_x —wyznacznik dla zmiennej X

W_y —wyznacznik dla zmiennej Y

B) gdy (($W=0$) i (($W_x \neq 0$) lub ($W_y \neq 0$))) układ sprzeczny

C) gdy (($W=0$) i ($W_x=0$) i ($W_y=0$)) układ ma nieskończenie wiele rozwiązań sposób obliczania wyznaczników

Wykonaj:

1) W zeszycie zapisz ile wynoszą elementy tablicy dla układu równań.

$$\begin{cases} nr_z_dziennika * x + miesiac_urodzenia * y = dzien_urodzenia \\ (nr_z_dziennika - 40) * x + liczba_liter_nazwska * y = liczba_liter_imienia \end{cases}$$

czyli A[0][0]=..... (wypisz wszystkie sześć elementów)

2) Przepisz teorię do zeszytu (obliczanie wyznaczników oraz przypadki rozwiązywalności)

Uwagi:

Ilość wierszy uzyskuje się przez **tablica.length**

a liczbę kolumn w danym wierszu uzyskuje się przez **tablica[numerWiersza].length**.

Wczytanie danych do tablicy dwuwymiarowej → do tego potrzebne są dwie pętle, jedna zagnieżdżona w drugiej.

```
for(int i = 0; i < tablica.length ; i++)
{
    for(int j = 0; j < tablica[i].length ; j++)
    {
        // tutaj dostępny jest element tablica[i][j]
    }
}
```

Program pisz etapami :

ETAP 1

Zapewnić wczytywanie danych do tablicy a oraz ich wypisywanie po wczytaniu z użyciem pętli.

Rozwiązanie Etapu 1:

1) Dokonaj dołączenia bibliotek umożliwiającej wczytywania danych z klawiatury

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

2) Utwórz klasę o nazwie Cztery_pierwsze_litery_Twojego_nazwiska

3) Utwórz funkcję główną:

```
public static void main(String[] args) throws NumberFormatException, IOException
```

4) utwórz obiekt tablicy o nazwie A_pierwsza_litera_nazwiska np.

```
float A[][] = new float [?][?];
```

w miejsce znaków zapytania wpisz odpowiednie liczby zgodne z treścią zadania.

5) utwórz obiekt czytania z klawiatury o nazwie czytaj_pierwsza_litera_nazwiska np.

```
BufferedReader czytaj = new BufferedReader(new InputStreamReader(System.in));
```

6) Wpisz wczytywanie danych do tablicy, pamiętaj o Twojej nazwie tablicy (innej niż w przykładzie poniżej). np.

```
for(int i = 0; i < A.length ; i++)
{
    for(int j = 0; j < A[i].length ; j++)
    {
        System.out.print("Podaj A[" + i + "][" + j + "] = ");
        A[i][j] = Float.parseFloat(czytaj.readLine());
    }
}
```


}
}

7)Wypisz dane z tablicy czytanej w podpunkcie 6)

ETAP 2

Obliczyć W_x , W_y , W , X , Y oraz wydrukować W_x , W_y , W , X i Y z dwoma miejscami po przecinku.

np. `String.format("%.2f",obwod)`

Potrzebne zmienne zadeklaruj jako `double`.

ETAP 3

Sprawdzenie warunków czy układ jest

- oznaczony
 - sprzeczny
 - ma nieskończenie wiele rozwiązań
- wraz z wyprowadzeniem stosownego komunikatu na monitorze

układy do testowania:

- oznaczony

wpisz swoje dane z układu

$$\begin{cases} nr_z_dziennika * x + miesiaki_urodzenia * y = dzien_urodzenia \\ (nr_z_dziennika - 40) * x + liczba_liter_nazwska * y = liczba_liter_imienia \end{cases}$$

- sprzeczny

$$\begin{cases} 1 * x + 2 * y = 2 \\ 1 * x + 2 * y = -5 \end{cases}$$

- ma nieskończenie wiele rozwiązań

$$\begin{cases} -3 * x + 4 * y = 2 \\ -3 * x + 4 * y = 2 \end{cases}$$

ETAP 4

Zapewnienie powtarzalności obliczeń

ETAP 5

Wypisanie na monitorze układu równań w postaci np.

$$\begin{aligned} 2 * X + 3 * Y &= 5 \\ -4 * X + 8 * Y &= 0 \end{aligned}$$

Wykonaj specyfikację problemu.

ZADANIE 19

1)Wykonaj praktycznie

- nazwa klasy `nazwisko_ucznia_z19` np. `Kowalski_z19`,

Napisz program obliczający wyznacznik 3x3 metodą Sarussa. Wczytaj macierz do tablicy trzy wiersze i trzy kolumny. Wydrukuj ją po wczytaniu wierszami oraz wartość wyznacznika. Znajdź w Internecie jak obliczamy wyznacznik 3x3 metodą Sarussa.

Wyjątki.

Definicja-wytlumaczenie.

Definicja

Wyjątek (ang. exceptions) to sytuacja nienormalna (powodująca błąd), która pojawia się w trakcie wykonania programu.

Wytlumaczenie

Mechanizm obsługi wyjątków w Javie umożliwia zaprogramowanie "wyjścia" z takich sytuacji krytycznych, dzięki czemu program nie zawiesi się po wystąpieniu błędu wykonując ciąg operacji obsługujących wyjątek. Wystąpienie sytuacji wyjątkowej przerywa "normalny" tok wykonania programu.

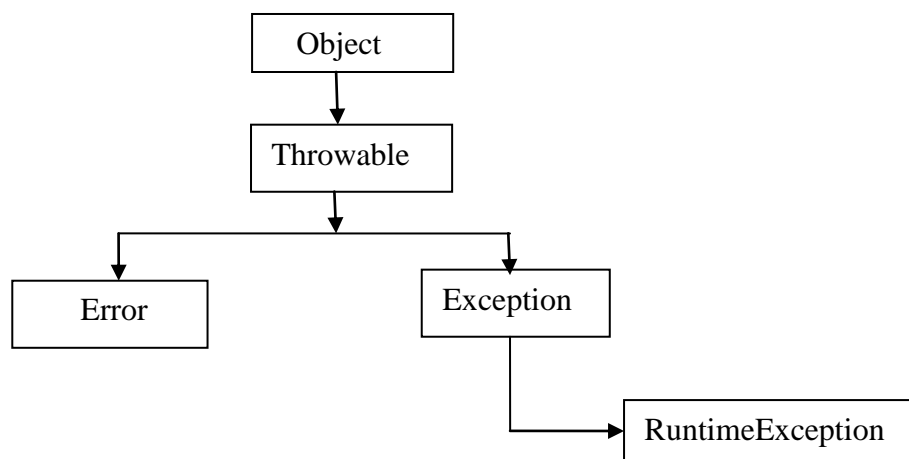
Instrukcja try ... catch...finally

```
try
{
    blok instrukcji mogący spowodować wyjątek
}
catch (TypWyjtku1 identyfikatorWyjtku1)
{
    obsługa wyjątku 1
}
catch (TypWyjtku2 identyfikatorWyjtku2)
{
    obsługa wyjątku 2
}
catch (TypWyjtkuN identyfikatorWyjtkuN)
{
    obsługa wyjątku n
}
finally
{
    czynności, które są wykonywane za każdym razem
}
```

Opis:

- **try** – otacza część programu(są to instrukcje, które wykonują program), którą chcemy monitorować na wypadek sygnalizacji błędów
- **catch** – w parze z **try**, wychwytuje określone wyjątki i obsługuje je i sterowanie przekazane do bloku instrukcji **catch**
- **throw** – sygnalizuje powstanie określonego wyjątku
- **throws** – Za pomocą instrukcji throws możemy przekazać obsługę wyjątku (jednego, kilku, wszystkich) poza metodę, w której wystąpią.
- **finally** – służy do wykonania kodu niezależnie od tego, czy wystąpił wyjątek, czy nie. Konieczne jest, kiedy trzeba przywrócić do pierwotnego stanu coś, co wymaga porządku, tak jak otwarty plik lub połączenie sieciowe.

Hierarchia dziedziczenia klas wyjątków



Throwable – obejmuje wszystkie wyjątki

Exception – wyjątki do wyłapania przez programy użytkowe

RuntimeException – definiowane automatycznie dla programów:

- dzielenie przez zero
- indeksowanie tablic
- itp.

TypWyjatk – wyjątki użytkownika

Error – nie do wyłapania przez programy użytkowe, błędy środowiska wykonawczego

Najczęściej wykorzystywane wyjątki to:

IOException - błąd we/wy, np. błąd przy wczytywaniu z klawiatury, czytanie za końcem pliku itp.;

ArithmeticException - niedozwolona operacja arytmetyczna, np. dzielenie przez zero;

ArrayIndexOutOfBoundsException - odwołanie do elementu spoza tablicy;

NumberFormatException - błąd konwersji na wartość liczbową;

NullPointerException - niepoprawne użycie wskaźnika null;

OutOfMemoryException - brak pamięci;

ClassCastException - rzutowanie obiektu na typ z innej gałęzi hierarchii klas, np. z Integer na String;

InterruptedException - wątek przerwał pracę innego wątku;

Przykład 15a

Temat: Demonstracja użycia wyjątków do obliczeń arytmetycznych (dzielenie przez zero)

Wykonaj:

1)Przepisz temat.

2)Zapisz w zeszycie informacje dotyczące wyjątków:

a)Definicja

b)Wytłumaczenie

c)Instrukcja try ... catch...finally→pamiętaj o wcięciach

d)Opis: try, Catch, throw, throws, finally

e)Hierarchia dziedziczenia klas wyjątków

3)Uruchom przykład. Zmień nazwę klasy na nazwisko_P15a np.Kowalski_P15a

a)z danymi: elementy=100 N=8 → przeczytaj komunikaty

b)z danymi: elementy=100 N=0 → przeczytaj komunikaty

4)Na podstawie komunikatów zapisz w zeszycie pytanie i uzupełnij miejsce kropek

Różnica komunikatów wynika z.....

```

.....
.....

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Przyklad_15a
{
    public static void main(String[] args) throws NumberFormatException, IOException
    {
        try
        {
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            int elementy,N;
            float wydajnosc;
            System.out.println("Umiem obliczać wydajność");
            System.out.println("obliczam w następujący sposób:   wydajnosc=elementy/N");
            System.out.print("Podaj ilość elementów elementy=");
            elementy = Integer.parseInt(reader.readLine());
            System.out.print("Podaj ilość pracowników N=");
            N = Integer.parseInt(reader.readLine());
            wydajnosc = elementy/N; // wystąpiło dzielenie przez zero
            System.out.println("wydajność="+wydajnosc+"\n");
        }

        catch (ArithmeticException e)
        {
            System.out.println("\n");
            System.out.println("Dzielenie przez zero");
            System.out.println("teraz komunikat systemowy");
            System.out.println(e); // komunikat systemowy
        }
        finally
        {
            System.out.println("\n");
            System.out.println("instrukcje finally");
            System.out.println("Jeśli nie wystąpił komunikat sytemowy to znaczy, że");
            System.out.println("wiesz, że musi być N<>0");
        }
    }
}

```

ZADANIE 19a

Napisz program:

1) Rezerwującą tablicę jednowymiarową o nazwie `tab_trzy_pierwsze_litery_nazwiska` np. `tab_kow` o wielkości `ilość_liter_imienia`. Jako wartości tablicy zapisz kolejne liczby pierwsze zaczynając od dwóch czyli `tab_kow[0]=2`;

jako wartość elementu `tab_kow[ilość_liter_imienia]` = kolejna liczba pierwsza.

Przypisanie wartości `tab_kow[ilość_liter_imienia]` = kolejna liczba pierwsza. Będzie powodować błąd.

Napisz program z użyciem wyjątku, który **`ArrayIndexOutOfBoundsException`** wykryje ten błąd i zapisz jaki błąd wystąpił. Użyj sekcji `try.... catch.....finally`. W każdej sekcji wpisz odpowiednie informacje i komentarze.

Wykonywanie programów w trybie Graficznym

Praca domowa:

W zeszycie udziel odpowiedzi na pytania: (przepisz pytania i poniżej treści pytania udziel odpowiedzi w zeszycie na pytanie). Odpowiedzi na pytania znajdziesz poniżej pytań (wiele stron).

- 1) Jak dzielone są programy trybu graficznego napisane w Java.
- 2) Opisz definicję aplikacji.
- 3) Opisz definicję apletu.
- 4) treść kodu HTML w, którym umieszczamy aplet.
- 5) w jaki sposób można uruchomić plik html, który jest plikiem uruchomieniowym apletu.
- 6) co nie może robić aplet.
- 7) zapisz ogólną składnia(strukturę) apletu.
- 8) Zapisz w zeszycie znaczenie (odpowiedzi masz w opisie działania programu /przykładu16):
 - klasa JApplet
 - pakiet java.awt
 - metoda paint.
 - clearRect,
 - drawString
 - getSize().width,
 - getSize().height
 - setBackground(),
 - setColor().

Przykładowe odpowiedzi w zeszycie ucznia:

klasa JApplet → klasa, która umożliwia stosowanie apletów na stronach WWW.

.....
.....

9) Co to są czcionki logiczne oraz wymień te czcionki?

10) Co to są czcionki fizyczne?

11) Opisz znaczenie trzech parametrów konstruktor klasy Font.

12) Zapisz w zeszycie znaczenie (odpowiedzi masz w komentarzach przykładu18):

- drawLine
- drawOval
- fillOval
- drawRect
- fillRect

Przykładowe odpowiedzi w zeszycie ucznia:

drawLine(int x1, int y1, int x2, int y2) → rysowanie linii od punktu (x1,y1) do punktu (x2,y2)

.....
.....

13) Zapisz w zeszycie znaczenie (odpowiedzi masz w przykładzie19):

- TextField pole
- pole=new TextField(5);
- pole.setText("0.00");
- add(pole1);
- s= pole1.getText();
- repaint();

Przykładowe odpowiedzi w zeszycie ucznia:

TextField pole; → utworzenie jednolinijkowego pola tekstowego

.....

.....
14)Zapisz w zeszycie znaczenie (odpowiedzi masz w przykładzie 20):

- getImage.
- getImage(URL url)
- getImage(URL url, String name)
- getDocumentBase()
- drawImage (img, wspX, wspC, this)

Przykładowe odpowiedzi w zeszycie ucznia:

getImage →
.....
.....

15)Co to jest przeciążenie metody. Jak rozróżniamy, którą metodę chcemy użyć w przypadku przeciążenia mimo tego, że mają te same nazwy.

(odpowiedź jest w przykładzie 21)

Odpowiedzi w dziale aplikacji.

16)Jak ułożone są klasy w pakietach,

17)Jak zbudowana jest hierarchia klas, podaj przykład z opisem,

18)Jakie są zalecenia odnośnie umieszczania klas w pakiecie zalecane przez Sun dla programistów, którzy piszą własne klasy?

18)Dlaczego w Java stworzono koncepcję kontenerów posiadających własne systemy zarządzania układem komponentów (layout managers)?

19)Wy tłumacz pojęcie panelu oraz kontener.

20)Zapisz jednym zadaniem wytłumaczenie do rozmieszczenia elementów panelu takich jak (do każdego jedno zdanie a do jednego rysunek) →FlowLayout, BorderLayout (do tego też rysunek), GridLayout, GridBagLayout, CardLayout

21)Opisz pole tekstowe jednolinijkowe- klasa JTextField (do czego służy, jak inicjujemy oraz jak można wprowadzić tekst początkowy)

22)Zapisz opis do klasy Font (składnia, styl, krój)

23)Do czego służy etykieta JLabel. Podaj oraz opisz trzy dowolne metody klasy JLabel.

24)Zapisz informacje o następujących instrukcjach graficznych:

- rysowanie wielokąta
- rysowanie wypełnionego wielokąta

25)Zapisz informacje o polu tekstowym JTextArea.

26)Zapisz informacje:

- W jakich bibliotekach można realizować przyciski Button (JButton)
- Deklaracja przycisku JButton
- Algorytm użycia przycisku JButton w programie

27)Zapisz informacje:

- znaczenie instrukcji dispose();
- definiowanie wielkości przycisku
- w jaki sposób definiujemy, który przycisk został wciśnięty
- definiowanie okna dialogowego (wraz z opcjami → wytłumaczenie)

Aplety.

Programy napisane w Jaview trybie graficznym mogą być:

- apletami (ang. applet) lub
- aplikacjami (ang. application).

Aplikacja jest programem samodzielnym uruchamianym z poziomu systemu operacyjnego (a dokładniej: maszyny wirtualnej Javy operującej na poziomie systemu operacyjnego) i tym samym ma pełny dostęp do zasobów udostępnianych przez system.

Aplet jest programem uruchamiany pod kontrolą innego programu, najczęściej przeglądarki internetowej, i ma dostęp jedynie do środowiska, które mu ten program udostępni. Inaczej aplet to program zagnieżdżony w innej aplikacji

Umieszczanie apletów w kodzie HTML

```
<html>
<body>
  <applet
    code = "PierwszyAplet.class"
    width = "300"
    height = "200">
  </applet>
</body>
</html>
```

Plikowi, w którym umieszczamy aplet nadajemy nazwę z rozszerzeniem html.

Plik html zawierające zagnieżdżone aplety może być uruchamiany na dwa sposoby:

- w przeglądarce,
- z użyciem komendy: `appletviewer nazwa_pliku.html`

Ogólna struktura apletu

Uwaga1:

Aplet musi być klasą publiczną i być rozszerzeniem klasy Applet pakietu java.awt lub klasą JApplet pakietu javax.swing

Uwaga2:

Aplety nie mogą:

- nie mogą uruchamiać aplikacji na komputerze użytkownika,
- nie mogą zapisywać i czytać plików na komputerze użytkownika,
- ściągać obrazów z innego serwera niż z serwera na którym są uruchamiane,
- Ładować bibliotek lokalnych i wywoływać metod natywnych,
- Łączyć się z innym hostem niż ten z którego zostały ściągnięte,
- Uruchamiać lokalnie polecenia,
- Okna otwierane przez aplet wyglądają inaczej niż te otwierane przez aplikację.

Ogólna składnia apletu.

```
import javax.swing.JApplet;
public class Aplet extends JApplet
```

```

{
    public Applet ( )
    {
        // Konstruktor apletu
    }
    public void init ( )
    {
        // Inicjalizacja apletu...
        // stosowana zamiast main()
    }
    public void start ( )
    {
        // Uruchomienie apletu...
    }
    public void stop ( )
    {
        // Zatrzymanie apletu...
    }
}

```

Uwaga:

W przypadku, gdyby miałyby zostać wykorzystana klasa Applet, dwa pierwsze wiersze należałoby zamienić na:

```

import java.awt.Applet;
public class Applet extends Applet

```

Przykład 16

Temat: Demonstracja działania apletu, pierwszy aplet, rysowanie, pisanie, kolory.

- 1) Zapisz temat w zeszycie.
- 2) Przeczytaj uwagi zapisane poniżej.
- 3) Wpisz poniższy aplet do edytora tekstu. Nazwij go: Nazwisko_ucznia.java np. Kowalski.java.
- 4) Dokonaj kompilacji. Pamiętaj o zgodności nazwy apletu z nazwą klasy publicznej.
- 5) Wykonaj plik html, w którym osadzisz aplet. Wielkość okna 300 na 400. Nazwa pliku html to index.html.
- 6) Uruchom aplet dwoma sposobami:
 - a) w przeglądarce → kliknij dwa razy szybko plik index.html
 - b) z użyciem komendy: *appletviewer nazwa_pliku.html* → z użyciem pliku BAT (patrz uwaga2, poniżej).

Uwaga1:

Gdy kopiujesz listingi z instrukcji a nie wpisujesz (listing pliku *.java oraz pliku *.html) pamiętaj o zmianie cudzysłowów na poprawnny oraz o zmianie nazwy klasy w pliku HTML.

Uwaga2:

Pliku BAT powinien mieć postać:

```
javac tutaj_twoje_nazwisko.java
pause
appletviewer index.html
```

Treść apletu:

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

public class PierwszyAplet extends JApplet
{
    public void paint (Graphics gDC)
    {
        setBackground(Color.lightGray);    //kolor tła
        gDC.setColor(Color.red);           // kolor pisania
        gDC.clearRect (0, 0, getSize( ).width, getSize( ).height);
        gDC.drawString ("Moje nazwisko, wpisz Swoje", 100, 50);
    }
}
```

Wy tłumaczenie działania **Apletu**:

Na początku importujemy klasę JApplet z pakietu javax.swing oraz pakiet java.awt.

Pakiet java.awt jest potrzebny, gdyż zawiera definicję klasy Graphics, dzięki której można wykonywać operacje graficzne. Aby utworzyć własny aplet, trzeba wyprowadzić klasę pochodną od JApplet, nasza klasa nazywa się po prostu PierwszyAplet.

Aplet będzie wyświetlany w przeglądarce i zajmie pewną część jej okna. Chcemy na powierzchni apletu wyświetlić napis, musimy więc w jakiś sposób uzyskać do niej dostęp. Pomaga nam w tym **metoda paint**. Jest ona automatycznie wywoływana za każdym razem, kiedy zaistnieje konieczność odrysowania powierzchni apletu. Metoda ta otrzymuje w argumencie wskaźnik do specjalnego obiektu udostępniającego metody pozwalające na wykonywanie operacji na powierzchni apletu.

Wywołujemy więc metodę **clearRect**, która wyczyści obszar okna apletu, a następnie **drawString** rysującą napis we wskazanych współrzędnych. W powyższym przypadku będzie to napis Moje nazwisko..... we współrzędnych x = 100 i y = 50. Metoda clearRect przyjmuje cztery argumenty określające prostokąt, który ma być wypełniony kolorem tła. Dwa pierwsze określają współrzędne lewego górnego rogu, a dwa kolejne — szerokość i wysokość.

Szerokość uzyskiwana jest przez wywołanie **getSize().width**, a wysokość **getSize().height**.

Uwaga:

W praktyce przed wywołaniem clearRect należałoby użyć metody **setColor** tak, aby na każdej platformie uruchomieniowej uzyskać taki sam kolor pisania. W różnych systemach domyślny kolor tła może być bowiem inny.

Koniec przykładu16

Przykład 17

Temat: Demonstarcja działania apletu, drugi aplet, rysowanie, pisanie, kolory, czcionki.

Wykonaj

- 1)Przepisz temat.
- 2)Wpisz poniższy aplet do edytora tekstu. Nazwij go: Nazwisko_ucznia_font.java np. Kowalski_font.java.
- 3)Dokonaj kompilacji. Pamiętaj o zgodności nazwy apletu z nazwą klasy publicznej.
- 4)Wykonaj plik html, w którym osadzisz aplet.
Wielkość okna: $300 + (\text{nr_z_dziennika} * 4)$ na $400 + (\text{nr_z_dziennia} * 3)$. Nazwa pliku html to index.html.
- 5)Uruchom aplet dwoma sposobami:
 - a) w przeglądarce → kliknij dwa razy szybko plik index.html
 - b) z użyciem komendy: *appletviewer nazwa_pliku.html* → z użyciem pliku BAT (patrz uwaga2, poniżej).

Uwaga:

Gdy kopiujesz listingi z instrukcji a nie wpisujesz(listing pliku *.java oraz pliku *.html) pamiętaj o zmianie cudzysłówów na poprawnw oraz o zmianie nazwy klasy w pliku HTML.

Temat przykładu:

Zastosowanie różnych wielkości czcionek, rodzajów, pogrubień. Ustaw wielkość wyświetlania okna na taka aby było widać wszystkie napisy związane z czcionkami (plik HTML).

Opis teoretyczna:

Czcionki w Javie dzielą się na dwa rodzaje:

czcionki logiczne to rodziny czcionek, które muszą być obsługiwane przez każde środowisko uruchomieniowe Javy. W środowisku Java jest dostępnych pięć czcionek logicznych:

- Serif,
- SansSerif,
- Monospaced,
- Dialog,
- DialogInput.

czcionki fizyczne czcionki fizyczne są zależne od systemu, na którym działa maszyna wirtualna

Działanie Apletu

W klasie Aplet przygotowujemy pięć pól klasy Font odpowiadających poszczególnym krojom pisma:

Serif,
SansSerif,
Monospaced,
Dialog,
DialogInput.

W metodzie init tworzymy pięć obiektów klasy Font i przypisujemy je przygotowanym polom.

Konstruktor klasy Font przyjmuje trzy parametry:

pierwszy z nich określa nazwę rodziny czcionek,

drugi styl czcionki,

trzeci jej wielkość.

Styl czcionki ustalamy, posługując się stałymi (zmiennie finalne) z klasy Font. Do dyspozycji mamy trzy różne wartości:

Font.BOLD — czcionka pogrubiona,

Font.ITALIC — czcionka pochylona,

Font.PLAIN — czcionka zwykła.

Do zmiany kroju wykorzystywanej czcionki stosujemy metodę o nazwie setFont. Przyjmuje ona jako argument obiekt klasy Font zawierający opis danej czcionki.

Uwaga:

W konstruktorze klasy Font można podać nazwę innej rodziny czcionek. Co się jednak stanie, jeśli dana czcionka nie istnieje w systemie? W takiej sytuacji zostanie wykorzystana czcionka domyślna. Czcionką domyślną jest Dialog.

Treść przykładu

```
import javax.swing.JApplet;
```

```
import java.awt.*;
```

```
public class twoje_nazwisko extends JApplet
```

```
{
```

```
    Font serif;
```

```
    Font sansSerif;
```

```
    Font monospaced;
```

```
    Font dialog;
```

```
    Font dialogInput;
```

```
    public void init()
```

```
    {
```

```
        serif = new Font("Serif", Font.BOLD, 24);
```

```
        sansSerif = new Font("SansSerif", Font.BOLD, 24);
```

```
        monospaced = new Font("Monospaced", Font.BOLD, 24);
```

```
        dialog = new Font("Dialog", Font.BOLD, 24);
```

```
        dialogInput = new Font("DialogInput", Font.BOLD, 24);
```

```
    }
```

```
    public void paint (Graphics gDC)
```

```
    {
```

```
        gDC.clearRect(0, 0, getSize().width, getSize().height);
```

```
        gDC.setFont(serif);
```

```
        gDC.drawString ("Czcionka Serif", 60, 40);
```

```
        gDC.setFont(sansSerif);
```

```
        gDC.drawString ("Czcionka SansSerif", 60, 80);
```

```
        gDC.setFont(monospaced);
```

```

gDC.drawString ("Czcionka Monospaced", 60, 120);
gDC.setFont(dialog);
gDC.drawString ("Czcionka Dialog", 60, 160);
gDC.setFont(dialogInput);
gDC.drawString ("Czcionka DialogInput", 60, 200);
}
}

```

Koniec przykładu 17

Numer z dziennika	Nazwa stałej (koloru)		Reprezentowany kolor
1 13 25	Color.black	Color.BLACK	Czarny
2 14 26	Color.blue	Color.BLUE	Niebieski
3 15 27	Color.darkGray	Color.DARK_GRAY	Ciemnoszary
4 16 28	Color.gray	Color.GRAY	Szary
5 17 29	Color.green	Color.GREEN	Zielony
6 18 30	Color.lightGray	Color.LIGHT_GRAY	Jasnoszary
7 19 31	Color.magenta	Color.MAGENTA	Karmazynowy
8 20 32	Color.orange	Color.ORANGE	Pomarańczowy
9 21 34	Color.pink	Color.PINK	Różowy
10 22 35	Color.red	Color.RED	Czerwony
11 23	Color.white	Color.WHITE	Biały
12 24	Color.yellow	Color.YELLOW	Żółty

Składowe RGB dla wybranych kolorów

Numer	Kolor	Składowa R	Składowa G	Składowa B
1	Beżowy	245	245	220
2	Biały	255	255	255
3	Błękitny	17	216	20
4	Brązowy	165	42	42
5	Czarny	0	0	0
6	Czerwony	255	0	0
7	Ciemnoczerwony	19	0	00
8	Ciemnoniebieski	0	0	19
9	Ciemnoszary	169	169	169
10	Ciemnozielony	0	100	0
11	Fiolet	28	10	28
12	Koralowy	255	127	80
13	Niebieski	0	0	255
14	Oliwkowy	128	128	0
15	Purpurowy	128	0	128
16	Srebrny	192	192	192
17	Stalowiebieski	70	10	180
18	Szary	128	128	128
19	Zielony	0	255	0
20	Żółtozielony	154	205	50
21	Żółty	255	255	0
22				

Zadanie 20

Temat. Pisanie tekstów oraz użycie kolorów i czcionek w Apletach.

Treść zadania:

Wykonaj aplet:

- zapisujący na tle zależnym od numeru w dzienniku, kolor to numer w dzienniku (patrz górna tabela kolorów powyżej),
np. `setBackground(Color.lightGray); //kolor tła`
- taką ilość linii, aby wyświetlać każdym wierszu w innym kolorze tekstu, dane techniczne komputera Twoich marzeń (dane pełne tak jak w reklamach).
np. `gDC.setColor(Color.red); // kolor pisania`
lub inaczej `gDC.setColor(New Color(200,100,70));`
- Napisy powinny być wraz z jednostkami np. Ghz.
- Użyj co najmniej 4 różne czcionki o różnych wielkościach.
- Wielkość okienka apletu tak aby mieściły się napisy.

Wykonaj:

1)Wykonaj program dokonaj kompilacji i uruchom z użyciem *appletviewer*

2)Przepisz listing programu do zeszytu.

Przykład 18

Temat: Demonstracja apletu, wykonującego obliczenia sumy odwrotności kwadratów dwóch liczb X Y.

Wykonaj

1)Przepisz temat.

2)Oblicz oraz zapisz w zeszycie sumę odwrotności kwadratów dwóch liczb X Y. Zastosuj kalkulator.

X=liczba liter nazwiska

Y=liczba liter imienia

$$SUMA = \frac{1}{X * X} + \frac{1}{Y * Y}$$

2)Wpisz poniższy aplet do edytora tekstu. Nazwij go: Nazwisko_ucznia_oblicz.java np. Kowalski_oblicz.java.

3)Dokonaj kompilacji. Pamiętaj o zgodności nazwy apletu z nazwą klasy publicznej.

4)Wykonaj plik html, w którym osadzisz aplet.

Wielkość okna: 300+(nr_z_dziennika*6) na 400+(nr_z_dziennia*5). Nazwa pliku html to index.html.

5)Uruchom aplet dwoma sposobami:

a) w przeglądarce→kliknij dwa razy szybko plik index.html

b) z użyciem komendy: *appletviewer nazwa_pliku.html*→z użyciem pliku BAT (patrz uwaga2, poniżej).


```

import java.awt.*;
import java.applet.*;

public class tutaj_twoje_nazwisko extends Applet {

    public tutaj_twoje_nazwisko() {

        TextField pole1; // utworzenie jednolinijkowego pola tekstowego
        TextField pole2;
        Font sansSerif; // definiowanie czcionki

        public void init() {
            sansSerif = new Font("SansSerif", Font.BOLD, 24);
            pole1 = new TextField();
            pole1.setBounds(100, 5, 60, 22);
            pole1.setText("0.00");
            pole2 = new TextField();
            pole2.setBounds(100, 35, 60, 22);
            pole2.setText("0.00");
            setLayout(null);
            add(pole1);
            add(pole2);
        }

        public void paint(Graphics g) {
            double liczba1, liczba2, suma;
            setBackground(Color.lightGray);
            g.drawString("X=", 51, 24);
            g.drawString("Y=", 51, 47);
            g.setColor(Color.black);
            g.drawString("Obliczam sume odrotnosci kwadratow dwoch liczb", 20, 72);
            g.drawString("zgodnie ze wzorem      SUMA = 1/(X*X)+1/(Y*Y)", 20, 92);
            // pisanie w okreslonym miejscu panelu
            g.setColor(Color.magenta);
            g.setFont(sansSerif);
            g.drawString("Podaj liczby i wcisnij ENTER:", 20, 120);
            String s1, s2, s3;
            s1 = pole1.getText(); // pobranie tekstu do zmiennej s1 z pola
            liczba1 = Float.parseFloat(s1);
            s2 = pole2.getText();
            liczba2 = Float.parseFloat(s2);
            suma = 1 / (liczba1 * liczba1) + 1 / (liczba2 * liczba2);
            s3 = String.format("%.4f", suma);
            g.setColor(Color.red);

            if ((liczba1 == 0) || (liczba2 == 0)) {
                g.drawString("suma_odwr= " + "wpisz liczby rozne od zera", 20, 162);
            } else {
                g.drawString("SUMA_odwr= " + s3, 20, 162);
            }
            this.setSize(600,200);
        }

        public boolean action(Event event, Object arg) {
            repaint(); // repaint wywołuje żądanie ponownego rysowania na panelu
            return true;
        }
    }
}

```

Opis instrukcji stosowanych w przykładzie

setLayout(null);

Możliwe jest, by kontener (miejsce gdzie umieszczamy elementu apletu/aplikacji) nie miał żadnego rozkładu elementów stosujemy wtedy setLayout(null).

W takim kontenerze pobługujemy się "absolutnym" pozycjonowaniem i wymiarowaniem komponentów np. za pomocą metody **setBounds(x, y, szerokość, wysokość)** w miejscu x y o szerokości i wysokości wielkości podawane w pikselach

TextField → to jednowierszowe pole tekstowe.

Klasa ta zawiera 4 wersje konstruktorów:

- * TextField() → Ta wersja konstruktora tworzy domyślnie pole tekstowe.
- * TextField(int szer) → Utworzone zostanie pole tekstowe o szerokości szer.
- * TextField(String str) → Przy użyciu tej wersji konstruktora w nowo powstałym komponencie będzie widoczny tekst przekazany w parametrze str.
- * TextField(String str, int szer) → Utworzone zostanie pole tekstowe o szerokości szer, i z napisem przekazanym w parametrze str.

Do **pobrania i ustawienia nowego** tekstu służą metody:

- *String getText()
- *void setText(String str)

Do **zaznaczenia tekstu** wykorzystujemy metodę:

- *void select(int startStr, int endStr)

Do **pobrania** aktualnie zaznaczonego tekstu używamy:

- *String getSelectedText()

Możliwość **modyfikacji** tekstu, możemy zmieniać za pomocą metody:

- *String setEditable(boolean edycja)

Jeśli prześlemy wartość true tekst w komponencie będzie można modyfikować.

Jeśli wartość ustawimy na false, nie będzie możliwości zmiany tekstu.

Do **sprawdzenia czy jest możliwość modyfikacji** używamy funkcji

- *isEditable(), która zwraca wartość typu boolean.

Możemy także włączyć funkcję **maskującą wpisywane znaki**:

- *setEchoChar(Char ch)

Gdzie jako parametr ch podstawiamy dowolny znak (np. '*');

Aby **wyłączyć maskowanie**, wywołujemy metodę:

- *char getEchoChar()

Do **obsługi zdarzeń** implementujemy:

- *interfejs ActionListener,
- który definiuje metodę actionPerformed(ActionEvent e).

Należy także wywołać metodę `addActionListener` klasy `TextField`, jako parametr podając klasę obsługującą zdarzenie.

Zadanie 21

Temat. Proste obliczenia w appletach.

Wykonaj:

- 1) Wykonaj program dokonaj kompilacji i uruchom z użyciem *appletviewer*
- 2) Przepisz listing programu do zeszytu.

Uwagi:

- Zmienne powinny być o znaczących nazwach np. jeśli jest to promień to powinno być to zmienna o nazwie „r”, jeśli jest to objętość to powinna to być zmienna „V”.
- Zastosuj dwa miejsca po przecinku (w instrukcji jest jak to zrobić).
- Wypisz jednostki np. cm^3 (jako centymetry sześciennne).
- Wczytywanie zmiennych: najpierw litera potem okienko np. X=

Numer w Tabelce	Treść zadania	Wynik
1	pole oraz obwód rombu po podaniu przekątnych,	d1=12 d2=16 P=96 cm^2 Obw= 40 cm
2	pole oraz obwód równoległoboku po podaniu dwóch boków oraz kąta między nim podany w stopniach,	a=5 b=4 alfa= 30 st Obw= 18 cm P= 10 cm^2
3	pole oraz obwód trójkąta po podaniu dwóch boków oraz kąta między nim podany w stopniach,	A=10 b=10 alfa=60 Obw=30 cm, c=10 cm P=43.3 cm^2
4	pole oraz objętość czworościanu foremnego po podaniu krawędzi bocznej	a=15 Pc=339 cm^2 (389) V=323 cm^3 (397)
5	oporu zastępczego po połączeniu dwóch oporników w połączeniu szeregowym i równoległych.	R1=1 ohm R2=9 ohm Rrow=0.9 ohm Rsz=10 ohm
6	pole oraz objętość walca po podaniu wysokości oraz promienia podstawy	r=10, h=15, V=4710,39 cm^3 PC=1570,79 cm^2
7	pole oraz objętość prostopadłościanu po podaniu jego trzech wymiarów.	a=3 b=4 c=5 V=60 cm^3 Pc=94 cm^2
8	pole oraz objętość prostopadłościanu prawidłowego trójkątnego po podaniu wysokości oraz krawędzi podstawy	a=10 , H=10 Pc=386,60 cm^2 V=433.01 cm^3
9	pędu i energii kinetycznej po podaniu prędkości i masy ciała	m= 10 kg v=5 m/s p=50 kg*m/s Ek=125 J
10	pole oraz obwód trójkąta równobocznego po podaniu boku a.	a=10 P=43.3 cm^2 Obw=10 cm
11	pole oraz objętość stożka po podaniu wysokości oraz promienia podstawy	r=10, h=15, V= PC=
12	pole oraz objętość ostrosłupa prawidłowego czworokątnego po podaniu wysokości i krawędzi	a=12 h=8 V=384 cm^3 Pc=336 cm^2

	podstawy	
13	pole oraz obwód koła po podaniu promienia.	$R=10$ $P=314,15 \text{ cm}^2$ $Obw=62,8 \text{ cm}$
14	pole oraz obwód kwadratu po podaniu jego przekątnej	
15	pole oraz obwód prostokąta po podaniu boków,	
16	pole oraz objętość prostopadłościanu prawidłowego sześciokątnego po podaniu krawędzi podstawy oraz wysokości.	$A=10 \text{ cm}$ $h=10 \text{ cm}$ $Pc=1119,62 \text{ cm}^2$ $V=2598,08 \text{ cm}^3$

Rysowanie figur geometrycznych w Apletach.

Przykład 19

Temat

Aplet rysujący prostokąt (składający się z czterech oddzielnych linii) z pojedynczym punktem w środku ciężkości oznaczonym literą S, zmień literę na pierwszą literę Twojego nazwiska. Tło zielone, napisy oraz rysunki czerwonym.

Wykonaj

- 1)Przepisz temat.
- 2)Uruchom aplet zmieniając nazwę klasy na Twoje_nazwisko.

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

```
public class Twoje_nazwisko extends JApplet  
{  
    public void paint (Graphics gDC)  
    {  
        setBackground(Color.green);  
        gDC.setColor(Color.red);  
        gDC.clearRect(0, 0, getSize().width, getSize().height);  
        gDC.drawLine(100, 40, 200, 40);  
        gDC.drawLine(100, 120, 200, 120);  
        gDC.drawLine(100, 40, 100, 120);  
        gDC.drawLine(200, 40, 200, 120);  
        gDC.drawLine(150, 80, 150, 80);  
        gDC.drawString ("S", 150, 100);  
    }  
}
```

drawLine(int x1,int y2, int x2, int y2)→ rysowanie linii

drawOval (int x, int y , int width, int height)→ rysowanie okręgu(elipsy) począwszy od punktu(x,y) o szerokości i wyokości, gdy szerokość=wysokość to okrąg.

fillOval (int x, int y , int width, int height)→ rysowanie wypełnionego koła (elipsy)

drawRect (int x, int y , int width, int height) →rysowanie Prostokąta począwszy od punktu(x,y) o szerokości i wyokości, gdy szerokość=wysokość to kwadrat

fillRect (int x, int y , int width,int height) →rysowanie wypełnienia Prostokąta

Zadanie 22

Temat

Użycie funkcji graficzne w apletach

Wykonaj

- 1)Przepisz temat.
- 2)Uruchom aplet zmieniając nazwę klasy na Twoje_nazwisko.
- 3)Przepisz listing programu.

Narysować trójkąt równoboczny o boku $200+5 \cdot (\text{numer z dziennika})$ pikseli oraz oznaczyć jego wierzchołki literami A B C.

W zeszycie narysuj układ współrzędnych (tylko jedna ćwiartka, pierwsza). Pamiętaj, że punkt (0,0) jest w górnym lewym rogu układu(ekranu) i strzałki osi będą skierowane na dół.

Narysuj trójkąt. Obok wierzchołków zapisz ich współrzędne. Do określenia współrzędnych wierzchołków możesz posłużyć się wzorem na wysokość w trójkącie równobocznym.

Treść poprzedniego przykładu wykorzystaj w zadaniu.

Wczytywanie grafiki

Wczytywanie obrazów graficznych umożliwia metoda klasy Applet o nazwie **getImage**.

Wczytuje ona grafikę w formacie GIF, JPG lub PNG.

Metoda getImage występuje w dwóch następujących wersjach:

- `getImage(URL url)`
- `getImage(URL url, String name)`

getDocumentBase()

metoda getDocumentBase zwraca obiekt URL(adres internetowy) wskazujący lokalizację dokumentu HTML (dokument w którym jest osadzony applet), w którym znajduje się applet,

getCodeBase()

metoda getCodeBase zwraca lokalizację internetową, w której znajduje się kod apletu.

drawImage (img, wspX, wspY, this);

wyświetla grafikę

argument **img** to referencja do obiektu klasy Image,

wspX → to współrzędna x,

wspY → to współrzędna y,

this to referencja (wskazanie) do obiektu implementującego interfejs ImageObserver

Przykład 20

Temat: Wyświetlanie grafiki w określonym miejscu wraz z opisem określonej czcionki i określonym rozmiarze w Aplecie.

Wykonaj

- 1)Przepisz temat.
- 2)Poproś nauczyciela arbuz.jpg
- 3)Uruchom aplet zmieniając nazwę klasy na Twoje_nazwisko.
 - a) w przeglądarce,
 - b) z użyciem komendy: appletviewer nazwa_pliku.html

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

public class Aplet extends JApplet
{
    Image obrazek;
    Font serif;

    public void init()
    {
        serif = new Font("Serif", Font.BOLD, 26);
        obrazek = getImage(getDocumentBase(), "arbuz.jpg");
    }

    public void paint(Graphics gDC)
    {
        setBackground(Color.lightGray); //kolor tła
        gDC.setColor(new Color(17,216,20)); // kolor pisania zdefiniowany przez RGB
        gDC.clearRect (0, 0, getSize( ).width, getSize( ).height);
        gDC.setFont(serif);
        gDC.drawString ("Arbuz to jest to", 100, 50);
        gDC.drawImage (obrazek, 120, 120, this);
    }
}
```

Opis działania przykładu.

Czwarty parametr metody **drawImage**, którym w przypadku apletu było wskazanie na obiekt tego apletu (wskazanie this).

Przed wszystkim musimy wiedzieć, co się dzieje w kolejnych fazach pracy takiego apletu.

W metodzie init wywołujemy **metodę getImage**, przekazując jej w argumencie lokalizację pliku. Ta metoda zwraca obiekt klasy Image niezależnie od tego, czy wskazany plik graficzny faktycznie istnieje, czy nie. Ładowanie danych rozpocznie się dopiero w momencie pierwszego wywołania metody drawImage.

Sama metoda drawImage działa natomiast w taki sposób, że po jej wywołaniu jest wyświetlana dostępna część obrazu (czyli albo nic, albo część obrazu, albo cały obraz) i metoda kończy działanie. Jeśli cały obraz był dostępny, jest zwracana wartość true, jeśli nie wartość false. Jeśli obraz nie był w pełni dostępny i zwrócona została wartość false, jest on

ładowany w tle. W trakcie tego ładowania, czyli napływania kolejnych danych z sieci, jest wywoływana metoda `imageUpdate` obiektu implementującego interfejs `ImageObserver`, który został przekazany jako czwarty argument metody `drawImage`.

Ponieważ klasa `JApplet` implementuje ten interfejs, możemy jej obiekt wykorzystać jako czwarty argument metody. Osiągamy wtedy sytuację, kiedy obiekt apletu jest informowany o postępach ładowania obrazu.

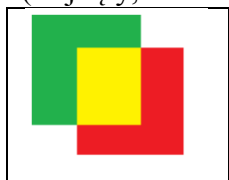
Zastosowana metoda `getImage` przyjmuje jako parametr obiekt klasy `URL` - w tym przypadku lokalizacja pliku HTML z kodem wywołującym applet. URL ten pobieramy za pomocą `getDocumentBase`. Drugi parametr to nazwa pliku graficznego. Metoda `drawImage` przyjmuje 4 parametry - pierwszy to referencja do obiektu typu `Image`, dwie kolejne to współrzędne, od których zacznie się wyświetlanie. Ostatni to `"this"`. Ostatni parametr musi być odnośnikiem do obiektu klasy implementującej interfejs `ImageObserver`.

Zadanie 23

Temat

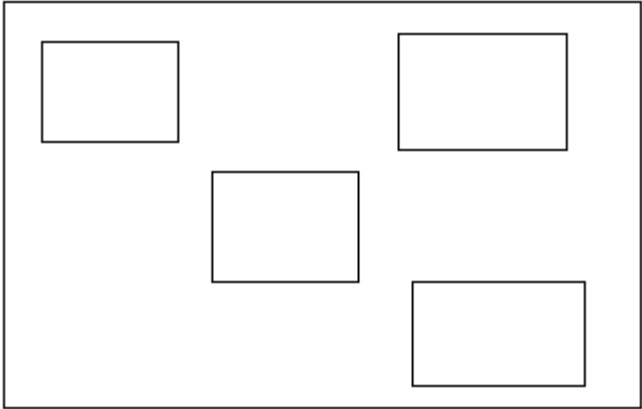
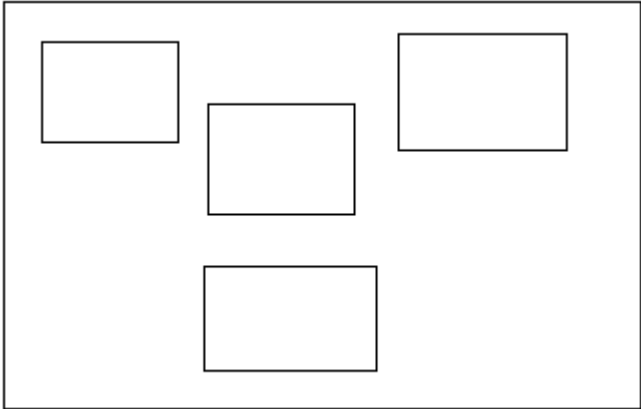
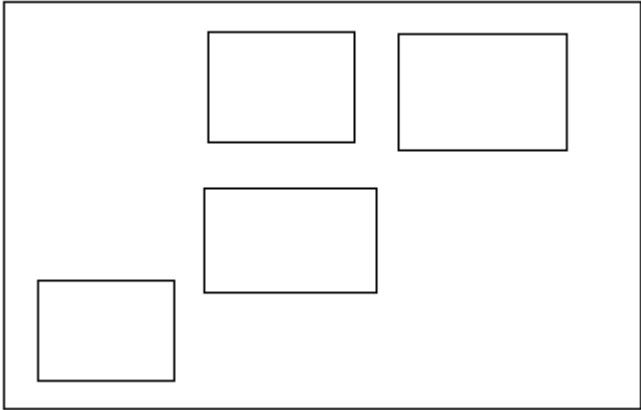
Wstawianie grafik w apletach oraz tekstów w postaci historyjki.

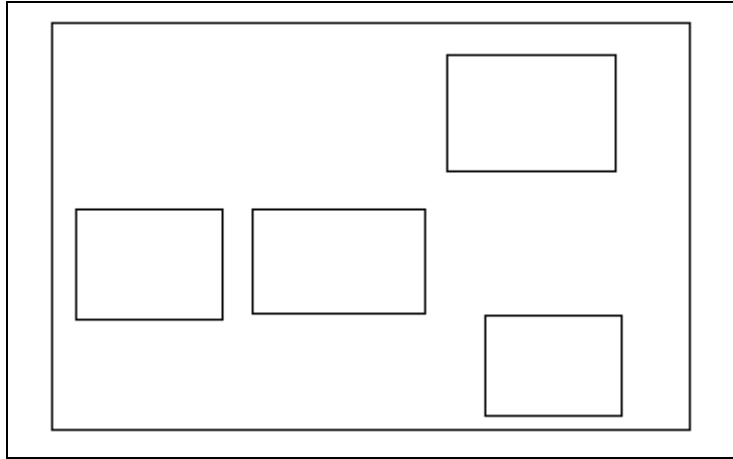
Zadanie powinno zawierać logo wykonane przez ucznia z użyciem instrukcji graficznych np. (trójkąty, kwadraty, okręgi, prostokąty, wypełnione)



Ułożenie grafik

Obliczenie	Który układ
$(\text{Numer_w_grupie}) \bmod 4 = 1$	Układ1
$(\text{Numer_w_grupie}) \bmod 4 = 2$	Układ2
$(\text{Numer_w_grupie}) \bmod 4 = 0$	Układ3
$(\text{Numer_w_grupie}) \bmod 4 = 3$	Układ4

Układ1

Układ3

Układ4

Układ2



Wykonaj

- 1)Przepisz temat.
- 2)Uruchom aplet zmieniając nazwę klasy na Twoje_nazwisko.
- 3)Przepisz listing programu.

Napisz program:

- wyświetlający, co najmniej cztery pliki graficzne z opisami odpowiadającymi grafice. Opis umieść pod grafiką. Całość ma tworzyć śmieszną przyzwoitą historyjkę. Pliki pobierz z Internetu. Opisy wymyśl sam.
- zastosuj kolor tła
- zastosuj cztery kolory czcionek oraz cztery różne ich wielkości
- Klasa zawierająca main ma mieć nazwę Nazwisko_ucznia_z23 (bez polskich liter).
- Wielkość okna $600+nr_z_dziennika$, $400+nr_z_dziennika$

Koniec zadania 23

Przykład 21

Temat: W trakcie ładowania obrazu na pasku stanu będzie wyświetlana informacja o procesie ładowania obrazka.

Wykonaj

- 1)Przepisz temat.
- 2)Uruchom aplet zmieniając nazwę klasy na Twoje_nazwisko.
- 3)Zapisz nagłówek metody, która przeciąża wraz z parametrami. Pod tym nagłówkiem zapisz, co oznaczają parametry tej metody wraz z wytłumaczeniem.

Przeciążenie metody.

Przeciążanie metod to definiowanie wielu metod o tej samej nazwie a różniących się listą argumentów.

Mamy, więc w naszej klasie zdefiniowanych kilka metod o tej samej nazwie, skąd, więc wiadomo, którą z nich wywołujemy w danej chwili? Naturalnie, na podstawie listy parametrów metody, (czyli ile oraz jakie parametry ma metoda).

Opis:

Wprowadzenie teoretyczne do uzyskania kontroli nad procesem ładowania obrazka

Gdybyśmy chcieli mieć możliwość kontroli procesu wczytywania i wyświetlania obrazu, należy **przeciążyć** metodę `imageUpdate` klasy `JApplet`.

```
import javax.swing.JApplet;
import java.awt.*;
import java.awt.image.*;
```

```
public class Aplet extends JApplet
{
    Image img;
    public void init()
    {
        img = getImage(getDocumentBase(), "image.jpg");
    }
    public void paint (Graphics gDC)
    {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawImage (img, 20, 20, this);
    }
}
```

```
public boolean imageUpdate(Image img, int flags, int x, int y, int width, int height)
{
    if ((flags & ImageObserver.ALLBITS) == 0)
    {
        showStatus ("Ładowanie obrazu...");
        return true;
    }
    else
    {

```

```

        showStatus ("Obraz załadowany");
        repaint();
        return false;
    }
}

```

Opis działania

Metoda **ipageUpdate** przy każdym wywołaniu otrzymuje cały zestaw argumentów, czyli:

ipg — referencję do rysowanego obrazu (jest to ważne, gdyż jednocześnie może być przecież ładowanych kilka obrazów),

x i y — współrzędne powierzchni apletu, w których rozpoczyna się obraz,

width i height — wysokość i szerokość obrazu oraz najważniejszy dla nas w tej chwili — **flags**.

Argument **flags** to wartość typu int, w której poszczególne bity informują o stanie ładowanego obrazu. Ich dokładne znaczenie można znaleźć w dokumentacji JDK w opisie interfejsu **IpageObserver**.

Najważniejszy dla nas jest **bit piaty** (o nazwie **ALLBITS**), którego ustawienie na 1 oznacza, że obraz został całkowicie załadowany i może zostać wyświetlony w ostatecznej formie. Do zbadania stanu tego bitu wykorzystujemy zdefiniowaną w klasie **IpageObserver** stałą (statyczna i finalne pole typu int) **IpageObserver.ALLBITS**. Jeśli wynikiem operacji bitowej **AND** między argumentem **flags** oraz stałą **ALLBITS** jest 0, oznacza to, że bit ten jest wyłączony, a zatem obraz nie jest załadowany, jeśli natomiast wynik tej operacji jest różny od 0, oznacza to, że bit jest włączony i dysponujemy pełnym obrazem.

Te właściwości wykorzystujemy zatem w metodzie **ipageUpdate**. Badamy wynik iloczynu logicznego **flags & IpageObserver.ALLBITS**. Kiedy jest on równy 0, wykonujemy metodę **showStatus** ustawiającą tekst na pasku stanu przeglądarki informujący, że obraz jest w trakcie ładowania; kiedy jest natomiast różny od 0, wyświetlamy napis, iż obraz został załadowany. W tym drugim przypadku należy dodatkowo odświeżyć ekran apletu, za co odpowiada metoda **repaint**.

Aplet z parametrami

Przykład 24

Wykonaj

- 1)Przepisz temat.
- 2)Zapisz w jaki sposób przekazujemy parametry ze strony WWW do Apletu
- 3)Zmień nazwę klasy na P24_Twoje_nazwisko.
- 4)Uruchom aplet.
 - a) w przeglądarce,
 - b) z użyciem komendy: appletviewer nazwa_pliku.html. Najwygodniej w Totalcommanderze wejdź do folderu z przykładem 24. W dolnym okienku wpisz komendę uruchomieniową apletu.

Temat: Przekazywanie parametrów ze strony WWW do Apletu.

Teoria→ jak przekazać parametry z WWW do Apletu:

Chcemy przekazać apletowi tekst, który będzie następnie wyświetlany z jego użyciem. Tworzymy zmienną(parametr) "tekst", do tego parametru wpisujemy z użyciem value wartość w tym przypadku tekst.

np. <param name="tekst" value="tutaj wpisz Twoje nazwisko">

Plik HTML

```
<applet
    code = "tutaj_twoje_nazwisko.class"
    width = "300"
    height = "100">
<param name="tekst" value="tutaj wpisz Twoje nazwisko">
</applet>
```

Plik apletu

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

public class tutaj_twoje_nazwisko extends JApplet
{
    private String tekst;
    public void init()
    {
        if((tekst = getParameter("tekst")) == null)
            tekst = "Nie został podany parametr: tekst";
    }
    public void paint (Graphics gDC)
    {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString (tekst, 60, 50);
    }
}
```

Koniec przykłady 24

Przykład 24a

Wykonaj

- 1)Przepisz temat.
- 2)Zmień nazwę klasy na P24a_Twoje_nazwisko.
- 3)Uruchom aplet
 - a) w przeglądarce,
 - b) z użyciem komendy: appletviewer nazwa_pliku.html

Temat:

Jak z pliku HTML przejąć parametry do apletu, w tym przypadku jest to tekst→Twoje nazwisko oraz kolor tła tutaj niebieski.

Plik HTML

```
<applet
  code = "tutaj_twoje_nazwisko.class"
  width = "400"
  height = "300">
  <param name="tekst" value="tutaj wpisz Twoje nazwisko">
  <param name="R" value="0">
  <param name="G" value="0">
  <param name="B" value="255">
</applet>
```

Plik apletu

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

public class tutaj_twoje_nazwisko extends JApplet
{
  private String tekst;
  public void init()
  {
    if((tekst = getParameter("tekst")) == null)
      tekst = "Nie został podany parametr: tekst";
  }
  public void paint (Graphics gDC)
  {
    gDC.setColor(Color.red);
    int R_I,G_I,B_I;
    // inicjowanie zmiennych
    R_I = Integer.parseInt(getParameter("R"));
    G_I = Integer.parseInt(getParameter("G"));
    B_I = Integer.parseInt(getParameter("B"));
    //konwersja na zmiennych typu Integer

    setBackground(new Color(R_I,G_I,B_I));
```

```

        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString (tekst, 60, 50);
    }
}

```

Zadanie 30

Wykonaj

- 1)Przepisz temat.
- 2)Nazwa klasy Z30_Twoje_nazwisko.
- 3)Uruchom aplet z30_Twoje_nazwisko.
 - a) w przeglądarce,
 - b) z użyciem komendy: appletviewer nazwa_pliku.html

Temat:

Applet zapisujący Twoje imię i nazwisko o określonym kolorze tła, kolorze liter i nazwisko w określonym miejscu ekranu. Wszystkie te informacje są przekazywane z użyciem parametrów z pliku HTML.

Treść zadania.

Parametry przekazane do apletu z pliku html.

W pliku html zdefiniuj parametry o nazwach:

naz_twoje_nazwisko np. naz_kowalski → do tego parametru wpisz Twoje imię i nazwisko

Uwaga: Kolor tła oraz liter będziesz definiował jako RGB (czyli przez trzy parametry).

tlo_twoje_nazwisko_R np. tlo_kowalski_R → do tego parametru wpisz nasycenie R dla tła (czerwone), podobnie zrób z G oraz B

litera_twoje_nazwisko np. litera_kowalski_R → do tego parametru wpisz nasycenie R dla kolor liter, podobnie zrób z G oraz B

x_twoje_nazwisko np. x_kowalski → do tego parametru wpisz współrzędną x umiejscowienia napisu

y_twoje_nazwisko np. y_kowalski → do tego parametru wpisz współrzędną y umiejscowienia napisu

uwaga:

Konieczna będzie konwersja zmiennej tekstowej (x i y) na zmienną integer.

```
int paramR=0;
```

```
// inicjowanie zmiennej o wartości zer
```

```
paramR = Integer.parseInt(getParameter("paramR"));
```

```
//konwersja na zmienną typu Integer
```

Kolory:

tła → to numer w dzienniku w tabeli kolorów pod przykładem 22 (kolorów jest 22)

liter → to numer dnia urodzenia dla dni mniejszych od 22, dla dni większych od 22 to 40-dzień urodzenia.

Zmienne x i y:

x → 50+numer_w_dzienniku,

y → 30+numer dnia urodzenia

Zadanie na przekazywanie danych obliczeniowych z HTML do Javy czyli coś obliczyć.

→ ułożyć zadanie na to oraz przykład.

Dźwięki i obsługa myszy → Interfejs `MouseListener`

Niektóre aplety wymagają reakcji na działania użytkownika, zwykle chodzi o zdarzenia związane z obsługą myszy. Jeśli aplet ma reagować na zmiany położenia kursora czy kliknięcia, może bezpośrednio implementować odpowiedni interfejs, bądź też korzystać z dodatkowego obiektu.

Interfejs `MouseListener` jest zdefiniowany w pakiecie `java.awt.event`, a zatem klasa, która będzie go implementowała, musi zawierać odpowiednią dyrektywę `import`. Jest on dostępny we wszystkich JDK, począwszy od wersji 1.1. Znajdują się w nim deklaracje pięciu metod zebranych w tabeli 8.4.

Każda klasa implementująca ten interfejs musi zatem zawierać definicję wszystkich wymienionych metod, nawet jeśli nie będzie ich wykorzystywała. Szkielet apletu będzie więc miał postać widoczną na listingu 8.18. Jak widzimy, mamy możliwość reagowania na pięć różnych zdarzeń opisanych w tabeli 8.4 i w komentarzach w zaprezentowanym kodzie.

Metody interfejsu `MouseListener`

Typ zwracany	Nazwa metody	Opis
Void	<code>mouseClicked(MouseEvent e)</code>	Metoda wywoływana po kliknięciu przyciskiem myszy.
Void	<code>mouseEntered(MouseEvent e)</code>	Metoda wywoływana, kiedy kursor myszy wejdzie w obszar komponentu.
Void	<code>mouseExited(MouseEvent e)</code>	Metoda wywoływana, kiedy kursor myszy opuści obszar komponentu.
Void	<code>mousePressed(MouseEvent e)</code>	Metoda wywoływana po naciśnięciu przycisku myszy.
Void	<code>mouseReleased(MouseEvent e)</code>	Metoda wywoływana po puszczeniu przycisku myszy.

Przykład 27

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

public class Aplet extends JApplet implements MouseListener
{
    public void mouseClicked(MouseEvent e)
    {
        //kod wykonywany po kliknięciu myszą
    }
    public void mouseEntered(MouseEvent e)
    {
        //kod wykonywany, kiedy kursor wejdzie w obszar komponentu
    }
    public void mouseExited(MouseEvent e)
    {
        //kod wykonywany, kiedy kursor opuści z obszaru komponentu
    }
}
```

```

public void mousePressed(MouseEvent e)
{
    //kod wykonywany, kiedy wciśnięty zostanie przycisk myszy
}
public void mouseReleased(MouseEvent e)
{
    //kod wykonywany, kiedy przycisk myszy zostanie zwolniony
}
}

```

Każda metoda otrzymuje jako argument obiekt klasy `MouseEvent` pozwalający określić rodzaj zdarzenia oraz współrzędne kursora.

Jeśli chcemy dowiedzieć się, który przycisk został wciśnięty, należy skorzystać z metody `getButton`, współrzędne natomiast otrzymamy, wywołując metody `getX` i `getY`. Metoda `getButton` zwraca wartość typu `int`, którą należy porównywać ze stałymi zdefiniowanymi w klasie `MouseEvent`:

```

MouseEvent.BUTTON1,
MouseEvent.BUTTON2,
MouseEvent.BUTTON3,
MouseEvent.NOBUTTON.

```

Pierwsze trzy określają numer przycisku, natomiast ostatnia informuje, że żaden przycisk podczas danego zdarzenia nie był wciśnięty.

Przykład 28

Przykładowy aplet, który wyświetla współrzędne ostatniego kliknięcia myszą oraz informację o tym, który przycisk został użyty.

```

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

public
class Aplet extends JApplet implements MouseListener
{
    String tekst = "";
    public void init()
    {
        addMouseListener(this);
    }
    public void paint (Graphics gDC)
    {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString(tekst, 20, 20);
    }
    public void mouseClicked(MouseEvent evt)
    {
        int button = evt.getButton();
    }
}

```

```

switch(button)
{
    case MouseEvent.BUTTON1 : tekst = "Przycisk 1, ";break;
    case MouseEvent.BUTTON2 : tekst = "Przycisk 2, ";break;
    case MouseEvent.BUTTON3 : tekst = "Przycisk 3 , ";break;
    default : tekst = "";
}
tekst += "współrzędne: x = " + evt.getX() + ", ";
tekst += "y = " + evt.getY();
repaint();
}
public void mouseEntered(MouseEvent evt){ }
public void mouseExited(MouseEvent evt){ }
public void mousePressed(MouseEvent evt){ }
public void mouseReleased(MouseEvent evt){ }
}

```

Wy tłumaczenie:

Ponieważ interesuje nas jedynie zdarzenie polegające na kliknięciu myszą, treść metod niezwiązanych z nim, czyli `mouseEntered`, `mouseExited`, `mousePressed`, `mouseReleased`, pozostaje pusta. Niemniej ich definicje muszą znajdować się w klasie `Aplet`, gdyż wymusza to interfejs `MouseListener`.

W metodzie `mouseClicked`, wywołując metodę **`getButton`**, odczytujemy kod naciśniętego przycisku i przypisujemy go zmiennej `button`. Następnie sprawdzamy wartość tej zmiennej za pomocą instrukcji wyboru `switch` i, w zależności od tego, czy jest to wartość `BUTTON1`, `BUTTON2` czy `BUTTON3`, przypisujemy odpowiedni ciąg znaków zmiennej `tekst`, która odpowiada za napis, jaki będzie wyświetlany na ekranie. W dalszej części kodu dodajemy do zmiennej `tekst` określenie współrzędnej `x` i współrzędnej `y`, w której nastąpiło kliknięcie. Wartości wymienionych współrzędnych odczytujemy dzięki metodom `getX` i `getY`.

Na końcu metody `mouseClicked` wywołujemy metodę **`repaint`**, która spowoduje odświeżenie ekranu. Odświeżenie ekranu wiąże się oczywiście z wywołaniem metody `paint`, w której wykorzystujemy znaną nam dobrze metodę `drawString` do wyświetlenia tekstu zawartego w polu `tekst`.

Bardzo ważny jest również fragment kodu, który wykonujemy w metodzie `init`. Otóż wywołujemy tam metodę **`addMouseListener`**, której w argumencie przekazujemy wskazanie do apletu. Takie wywołanie oznacza, że wszystkie zdarzenia związane z obsługą myszy, określone przez interfejs `MouseListener`, będą obsługiwane przez obiekt przekazany tej metodzie jako argument. Ponieważ argumentem jest sam obiekt apletu (choć może być to obiekt dowolnej klasy implementującej interfejs `MouseListener`), to w tym przypadku informujemy po prostu maszynę wirtualną, że nasz aplet samodzielnie będzie obsługiwał zdarzenia związane z myszą.

O wywołaniu metody `addMouseListener` koniecznie musimy pamiętać, gdyż jeśli jej zabraknie, kompilator nie zgłosi żadnego błędu, a program po prostu nie będzie działał. Jest to błąd stosunkowo często popełniany przez początkujących programistów.

Skoro jednak metoda `addActionListener` może przekazać obsługę zdarzeń dowolnemu obiektowi implementującemu interfejs `MouseListener`, zobaczmy, jak by to wyglądało w praktyce. Napiszemy dodatkową klasę pakietową współpracującą z klasą `Aplet` i odpowiedzialną za obsługę myszy. Aby nie komplikować kodu, jej zadaniem będzie

wyświetlenie na konsoli współrzędnych ostatniego kliknięcia. Kod obu klas został zaprezentowany

Przykład 29

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

public
class Aplet extends JApplet
{
    public void init()
    {
        addMouseListener(new MyMouseListener());
    }
}

class
MyMouseListener implements MouseListener
{
    public void mouseClicked(MouseEvent evt)
    {
        String tekst = "";
        int button = evt.getButton();
        switch(button)
        {
            case MouseEvent.BUTTON1 : tekst = "Przycisk 1, ";break;
            case MouseEvent.BUTTON2 : tekst = "Przycisk 2, ";break;
            case MouseEvent.BUTTON : tekst = "Przycisk , ";break;
            default : tekst = "";
        }
        tekst += "współrzędne: x = " + evt.getX() + ", ";
        tekst += "y = " + evt.getY();
        System.out.println(tekst);
    }
    public void mouseEntered(MouseEvent evt){ }
    public void mouseExited(MouseEvent evt){ }
    public void mousePressed(MouseEvent evt){ }
    public void mouseReleased(MouseEvent evt){ }
}
```

Opis

Klasa Aplet nie implementuje w tej chwili interfejsu MouseListener, gdyż obsługa myszy jest przekazywana innej klasie. Pozostała w niej jedynie metoda init, w której wywołujemy metodę addActionListener. Argumentem przekazanym addActionListener jest nowy obiekt klasy MyMouseListener. Oznacza to, że aplet ma reagować na zdarzenia myszy, ale ich obsługa została przekazana obiektowi klasy MyMouseListener.

Klasa `MyMouseListener` jest klasą pakietową, jest zatem zdefiniowana w tym samym pliku, co klasa `Aplet`. Implementuje ona oczywiście interfejs `MouseListener`, inaczej obiekt tej klasy nie mógłby być argumentem metody `addActionListener`. Wewnątrz klasy `MyMouseListener` zostały zdefiniowane metody z interfejsu, jednak kod wykonywalny zawiera jedynie metodę `mouseClicked`. Jej treść jest bardzo podobna do kodu metody `mouseClicked` z poprzedniego przykładu. Jedyną różnicą jest to, że zmienna `tekst` jest zdefiniowana wewnątrz tej metody i zamiast metody `repaint` jest wykonywana instrukcja `System.out.println` wyświetlająca na konsoli współrzędne kliknięcia. Uruchomienie takiego apletu spowoduje, że współrzędne kliknięcia będą się pojawiały na konsoli, Nie możemy tym razem wyświetlać tekstu bezpośrednio w obszarze apletu, gdyż klasa `MyMouseListener` nie ma do niego dostępu. Jak go uzyskać? Można by na przykład przekazać referencję do obiektu apletu w konstruktorze klasy `MyMouseListener`

Interfejs `MouseMotionListener`

Interfejs `MouseMotionListener` pozwala na obsługiwanie zdarzeń związanych z ruchem myszy.

Definiuje on dwie metody.

- pierwsza z nich jest wywoływana, kiedy przycisk myszy został wciśnięty i mysz się porusza,
- druga przy każdym ruchu myszy bez wciśniętego przycisku.

Aplet, który będzie wyświetlał aktualne współrzędne położenia kursora myszy.

Metody interfejsu `MouseMotionListener`

Typ zwracany	Nazwa metody	Opis
Void	<code>mouseDragged(MouseEvent e)</code>	Metoda wywoływana podczas ruchu myszy, kiedy wciśnięty jest jeden z klawiszy.
Void	<code>mouseMoved(MouseEvent e)</code>	Metoda wywoływana przy każdym ruchu myszy, o ile nie jest wciśnięty żaden klawisz.

Przykład 30

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

public
class Aplet extends JApplet implements MouseMotionListener
{
    String tekst = "";
    public void init()
    {
        addMouseMotionListener(this);
    }
    public void paint (Graphics gDC)
```

```

{
    gDC.clearRect(0, 0, getSize().width, getSize().height);
    gDC.drawString(tekst, 20, 20);
}
public void mouseMoved(MouseEvent evt)
{
    tekst = "zdarzenie mouseMoved, ";
    tekst += "współrzędne: x = " + evt.getX() + ", ";
    tekst += "y = " + evt.getY();
    repaint();
}
public void mouseDragged(MouseEvent evt)
{
    tekst = "zdarzenie mouseDragged, ";
    tekst += "współrzędne: x = " + evt.getX() + ", ";
    tekst += "y = " + evt.getY();
    repaint();
}
}

```

Sposób postępowania jest tu identyczny jak w przypadku interfejsu `MouseListener`. Klasa `Aplet` implementuje interfejs `MouseMotionListener`, zawiera zatem metody `mouseMoved` i `mouseDragged`. W metodzie `init` jest wywoływana metoda `addMouseMotionListener` (analogicznie jak we wcześniejszych przykładach `addMouseListener`), dzięki czemu wszystkie zdarzenia związane z poruszaniem myszy będą obsługiwane przez klasę `Aplet`. W metodach `mouseMoved` oraz `mouseDragged` odczytujemy współrzędne kursora myszy dzięki funkcjom `getX` i `getY`, przygotowujemy treść pola tekst oraz wywołujemy metodę `repaint` odświeżającą ekran. W metodzie `paint` wyświetlamy zawartość pola tekst na ekranie.

Dodatkowe parametry zdarzenia

Niekiedy przy przetwarzaniu zdarzeń związanych z obsługą myszy zachodzi potrzeba sprawdzenia stanu klawiszy specjalnych `Alt`, `Shift` lub `Ctrl`. Z taką sytuacją mamy do czynienia np. wtedy, kiedy program ma inaczej reagować, kiedy kliknięcie nastąpiło z równoczesnym wciśnięciem jednego z wymienionych klawiszy. Musi zatem istnieć sposób pozwalający na sprawdzenie, czy mamy do czynienia z taką specjalną sytuacją. Tym sposobem jest dokładniejsze zbadanie obiektu klasy `MouseEvent`, który jest przekazywany funkcji obsługującej każde zdarzenie związane z obsługą myszy.

Klasa `MouseEvent` (a dokładniej klasa `InputEvent`, z której `MouseEvent` dziedziczy) udostępnia metodę o nazwie `getModifiers`, zwracającą wartość typu `int`, której poszczególne bity określają dodatkowe parametry zdarzenia. Stan tych bitów badamy poprzez porównanie z jedną ze stałych6 zdefiniowanych w klasie `MouseEvent`. W sumie jest dostępnych kilkadziesiąt stałych, w większości odziedziczonych z klas bazowych, ich opis można znaleźć w dokumentacji JDK. Dla nas interesujące są trzy wartości:

- `MouseEvent.SHIFT_DOWN_MASK`,
- `MouseEvent.ALT_DOWN_MASK`,
- `MouseEvent.CTRL_DOWN_MASK`.

Pierwsza z nich oznacza, że został wciśnięty klawisz `Shift`, druga, że został wciśnięty klawisz `Alt`, a trzecia, że został wciśnięty klawisz `Ctrl`. Porównania z wartością zwróconą przez

getModifiers dokonujemy przez wykonanie operacji bitowej AND, czyli iloczynu bitowego. Jeśli więc wynikiem operacji: getModifiers() & MouseEvent.SHIFT_DOWN jest wartość 0, oznacza to, że klawisz Shift nie był wciśnięty, a jeśli wartość tej operacji jest różna od 0, oznacza to, że Shift był wciśnięty.

Przykładowy aplet wykorzystujący opisaną technikę do stwierdzenia, które z klawiszy funkcyjnych były wciśnięte podczas przesuwania kursora myszy.

Przykład 31

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

public
class Aplet extends JApplet implements MouseMotionListener
{
    String tekst = "";
    public void init()
    {
        addMouseMotionListener(this);
    }
    public void paint (Graphics gDC)
    {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString(tekst, 20, 20);
    }
    public void mouseMoved(MouseEvent evt)
    {
        tekst = "Wciśnięte klawisze [";
        int modifiers = evt.getModifiersEx();
        if((modifiers & MouseEvent.SHIFT_DOWN_MASK) != 0)
        {
            tekst += " SHIFT ";
        }
        if((modifiers & MouseEvent.ALT_DOWN_MASK) != 0)
        {
            tekst += " ALT ";
        }
        if((modifiers & MouseEvent.CTRL_DOWN_MASK) != 0)
        {
            tekst += " CTRL ";
        }
        tekst += "], ";
        tekst += "współrzędne: x = " + evt.getX() + ", ";
        tekst += "y = " + evt.getY();
        repaint();
    }
    public void mouseDragged(MouseEvent evt)
    {

```

```
}  
}
```

Dźwięki

Pisane przez nas aplety możemy wyposażyć w możliwość odtwarzania dźwięków. Java obsługuje standardowo kilka formatów plików dźwiękowych, są to AU, AIFF, WAVE oraz MIDI. W klasie JApplet została zdefiniowana metoda play, która pobiera i odtwarza klip dźwiękowy.

Występuje ona w dwóch przeciążonych wersjach:

- play(URL url)
- play(URL url, String name).

Pierwsza z nich przyjmuje jako argument obiekt klasy URL bezpośrednio wskazujący na plik dźwiękowy, druga wymaga podania argumentu klasy URL wskazującego na umiejscowienie pliku (np. `http://host.domena/java/sound/`) i drugiego określającego nazwę pliku. Jeśli plik znajduje się w strukturze katalogów naszego serwera, wygodniejsze może być użycie drugiej postaci konstruktora, podobnie jak w przypadku metody `getImage`

Przykład 32

```
import javax.swing.JApplet;  
  
public  
class Aplet extends JApplet  
{  
    public void start()  
    {  
        play (getDocumentBase(), "ding.au");  
    }  
}
```

Drugim sposobem na odtwarzanie dźwięku jest wykorzystanie interfejsu `AudioClip`. Interfejs ten definiuje trzy metody: `start`, `stop` i `loop`. Metoda `start` rozpoczyna odtwarzanie dźwięku, `stop` kończy odtwarzanie, natomiast `loop` rozpoczyna odtwarzanie dźwięku w pętli. Ponieważ `AudioClip` został zdefiniowany w pakiecie `java.applet`, tym razem jako klasę apletu wykorzystamy `Applet` (zamiast `JApplet`). Obiekt implementujący interfejs `AudioClip` otrzymamy, wywołując metodę `getAudioClip` klasy `Applet`. Metoda ta występuje w dwóch przeciążonych wersjach:

- `getAudioClip(URL url)`
- `getAudioClip(URL url, String name)`

Znaczenie argumentów jest takie samo, jak w przypadku opisanej wyżej metody `play`.

Po uruchomieniu apletu rozpoczyna się odtwarzanie dźwięku, a przy kończeniu jego pracy odtwarzanie jest zatrzymywane.

Przykład 33

```
import java.applet.*;  
  
public
```



```

class Aplet extends Applet
{
    AudioClip audioClip;
    public void init() {
        audioClip = getAudioClip(getDocumentBase(), "ding.au");
    }
    public void start()
    {
        audioClip.loop();
    }
    public void stop()
    {
        audioClip.stop();
    }
}

```

Przykład 34

Znacznie ciekawsze byłoby jednak połączenie możliwości odtwarzania dźwięków oraz reakcji na zdarzenia związane z obsługą myszy. Wykorzystanie możliwości, jakie dają interfejsy `MouseListener` oraz `AudioClip`, pozwala na napisanie apletu, który będzie np. odtwarzał plik dźwiękowy, kiedy użytkownik kliknie myszą.

```

import java.applet.*;
import java.awt.event.*;

public
class Aplet extends Applet implements MouseListener
{
    AudioClip audioClip;
    public void init()
    {
        addMouseListener(this);
        audioClip = getAudioClip(getDocumentBase(), "ding.au");
    }
    public void mouseClicked(MouseEvent evt)
    {
        audioClip.play();
    }
    public void mouseEntered(MouseEvent evt){ }
    public void mouseExited(MouseEvent evt){ }
    public void mousePressed(MouseEvent evt){ }
    public void mouseReleased(MouseEvent evt){ }
}

```

Opis przykładu

Klasa `Aplet` implementuje interfejs `MouseListener`, a zatem zawiera definicje wszystkich jego metod. Wykorzystujemy jednak jedynie metodę `mouseClicked`, która będzie wywoływana po każdym kliknięciu myszą. Rozpoczynamy w niej odtwarzanie dźwięku poprzez wywołanie metody `play` obiektu `audioClip`. Obiekt wskazywany przez pole `audioClip` uzyskujemy w

metodzie `init` przez wywołanie metody `getAudioClip` klasy `Applet`, dokładnie w taki sam sposób, jak w poprzednim przykładzie. Nie zapominamy również o wywołaniu metody `addMouseListener`, bez której aplet nie będzie reagował na kliknięcia.

Ćwiczenia do samodzielnego wykonania

Ćwiczenie 40.1.

Zmień kod apletu z listingu 8.19 w taki sposób, aby reagował nie na kliknięcia, ale na samo naciśnięcie przycisku myszy.

Ćwiczenie 40.2.

Napisz aplet, w którym obsługa ruchów myszy będzie realizowana przez oddzielną klasę `MyMouseMotionListener`. Przy każdym ruchu myszy wyświetl współrzędne kursora myszy na konsoli.

Ćwiczenie 40.3.

Napisz aplet, który będzie odtwarzał plik dźwiękowy, kiedy użytkownik zbliży kursor myszy na mniej niż 10 pikseli od brzegów powierzchni apletu. Wysokość oraz szerokość obszaru apletu można uzyskać, wywołując metody `getWidth` oraz `getHeight` klasy `Applet`.

Ćwiczenie 40.4.

Napisz aplet, który przy kliknięciu będzie odtwarzał dźwięki. Odtwarzanie powinno być realizowane przez pakietową klasę `MyAudioClip` implementującą interfejs `AudioClip`.

Aplikacje

Lekcja 41. Tworzenie aplikacji

Na początku rozdziału 8. poznaliśmy różnicę między aplikacją i apletem, wiemy, że aplikacja potrzebuje do uruchomienia jedynie maszyny wirtualnej, a aplet jest programem wbudowanym, zagnieżdżonym w innym programie, najczęściej w przeglądarce internetowej. Wszystkie programy, które powstawały w rozdziałach 1. – 7., były właśnie aplikacjami, pracującymi jednak w trybie tekstowym. W tej lekcji zobaczymy, w jaki sposób tworzy się aplikacje pracujące w trybie graficznym, czyli popularne aplikacje okienkowe.

Aplikacje trybu graficznego.

Pakiety

Java nie jest monolitem, lecz składa się z szeregu klas definiujących obiekty różnego typu. Dla przejrzystości klasy te pogrupowane są w hierarchicznie ułożone pakiety.

Każdy pakiet grupuje klasy związane z pewnym szerokim zakresem zastosowań języka np.

- java.io (klasy wejścia-wyjścia),
- java.util.prefs (klasy użytkowe do obsługi preferencji),
- java.awt (system obsługi trybu graficznego),
- import javax.swing. (system obsługi trybu graficznego)

Hierarchię klas oddają nazwy pakietów, które skonstruowane są podobnie jak ścieżki dostępu do plików. Na przykład klasa Preferences znajdująca się w pakiecie java.util.prefs ma pełną nazwę: java.util.prefs.Preferences, co oznacza:

- java – pakiet należy do zestawu standardowych pakietów Javy,
- util – to różnego typu klasy użytkowe (pomocnicze) głównie organizujące obsługę różnego typu struktur danych,
- prefs – system obsługi preferencji w sposób niezależny od platformy, w którym preferencje systemowe i użytkownika są składowane w postaci hierarchicznego rejestru,
- Preferences – konkretna nazwa klasy.

Dzięki takiemu systemowi nazwy klas są niepowtarzalne, co pozwala uniknąć niejednoznaczności (np. czy chodzi o klasę List implementującą strukturę listy danych czy o List implementującą graficzną listę wyświetlaną w okienku).

Wszystkie klasy pisane przez programistów niezależnych powinny być umieszczane w innych hierarchiach. Firma Sun często zaleca, by w nazewnictwie klas niestandardowych przed właściwą nazwą pakietu stosować odwróconą nazwę domeny internetowej autora pakietu. Na przykład narzędzie Ant znajduje się w pakiecie org.apache.ant, co zapobiega konfliktom nazw z pakietami innych autorów, którzy również chcieliby nazwać swój pakiet Ant.

Domyślnie klasy pakietu nie są możliwe do użycia poza nim. Stąd nie występują konflikty nazw klas przy imporcie różnych pakietów. Klasa pakietu staje się publiczną przy deklaracji `public class Foo`.

Główne okno programu tworzymy z użyciem klasy `JFrame`.

Aby móc korzystać z tej klasy musimy także załadować pakiet **swing**. Należy więc w programie umieścić instrukcje:

```
import javax.swing.*;
```

W klasie `JFrame` możemy posługiwać się metodami:

```
JFrame okno;
```

```
JPanel pan;
```

```
boolean war;
```

```
int w,h,x,y;
```

```
okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

powoduje wyjście z programu po zamknięciu okna

okno.setVisible(boolean flag)

jeśli flag=true to okno jest widoczne

okno.setTitle("nazwa okna")

informacja wyświetlająca się na górnej belce okna

okno.setResizable(boolean flag)

jeśli flag=true to okno można skalować przy użyciu myszki

okno.setSize(int w, int h)

ustala rozmiar okna (w- szerokość, h- wysokość w pikselach)

okno.setLocation(int x, int y)

ustalane jest położenie lewego górnego rogu okna

okno.setContentPane(pan)

ładuje panel dla naszego okna(wyjaśnienie czym jest panel znajduje się w kolejnym podrozdziale)

okno.pack() po umieszczeniu składników w panelu i załadowaniu - panel

jest uruchamiany (wyjaśnienie w kolejnym podrozdziale)

ZADANIE 23a

Temat: Wykonanie oraz rozmieszczenie okna w aplikacji.

Napisz program na podstawie przykładu(patrz poniżej), który utworzy okno o ustalonych wymiarach geometrycznych, których nie można zmieniać.

Zmień przykład tak, aby:

- Nazwa klasy → class Okienko_nazwisko_ucznia_z23a;
- w tytule okna wyświetliło się Twoje nazwisko
- rozmiar okna (400+nr_z_dziennika, 200+nr_z_dziennika)
- umieszczenie okna (200+nr_z_dziennika, 200+nr_z_dziennika)

```
import javax.swing.*;

class Okienko_nazwisko_ucznia_z23a // wpisz taką nazwę klasy
{
    public static void main(String[] args)
    {
        JFrame okno=new JFrame();
        //wlasnosci
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setVisible(true);
        //wlasnosci - rozmiar
        okno.setResizable(false);
        okno.setSize(400,200);
        okno.setLocation(200,200);
        //informacja wyswietlana w naglowku
        okno.setTitle("tutaj wpisz Twoje nazwisko");
    }
}
```

ZADANIE 24

Zmień zadanie/przykład poprzednie tak, aby:

- Nazwa klasy → `class Okienko_nazwisko_ucznia_z24;`
- można było skalować myszką okno (sprawdź czy można zmienić wielkość okna myszką),
- zmień kolor tła na zielony poprzez:

a) dołącz pakiet **java.awt**, użyj polecenia `import`,

b) wpisz wywołanie metody zmiany koloru poprzez

`nazwa_obiektu.getContentPane().setBackground(Color.nazwa_koloru_ang);`
nazwa koloru małymi literami.

Koniec zadania 24.

Zawartość okna - klasa JPanel, definicja kontenera.

Program przedstawiony powyżej tworzy tylko okno.

Okno to można podzielić na dwie części:

górna, w której widnieje jakaś informacja (kolor niebieski)

dolna w kolorze szarym, dolna część służy do umieszczania składników takich jak:

- przyciski,
- pola tekstowe,
- czy obszarów do rysowania.

Aby móc zarządzać dolną częścią okna należy posłużyć się "kontenerem" (content pane).

Kontener jest to obiekt, w którym możemy umieszczać inne obiekty oraz inne kontenery. Jest to wirtualne pudełko, do którego możemy wsadzać inne wirtualne przedmioty. Panel może być kontenerem.

Do wypełniania okna służy obiekt panel. Staje się on kontenerem.

Składania polecenia tworzącego potrzebny obiekt jest następująca:

```
JPanel pan=new JPanel();
```

Metody dostępne dla obiektu klasy JPanel:

metoda opis

int x,y;

JPanel pan=new JPanel();

Dimension roz=new Dimension(x,y);

Color kolor;

`Color(R,G,B)` → klasa, służy do zmiany kolorów używając schematu RGB.

`pan.setLayout(new Rozmieszczenie)`

sposób rozmieszczania składników w panelu określa

`Rozmieszczenie(`

np.: `Layout=FlowLayout(),`

`BorderLayout(),`

`GridLayout(),`

`BoxLayout(itp.)`

Rozmieszczenie komponentów (panelu).

Java jest stworzona do konstrukcji aplikacji wykonywalnych na dowolnych platformach docelowych. Aplikacje te powinny zachowywać się identycznie na każdym komputerze, pod dowolnym systemem operacyjnym (wyposażonym oczywiście w wirtualną

maszynę javy). Takie założenie właściwie wyklucza zwykłe podejście do budowania interfejsu użytkownika w postaci układu elementów o z góry określonych pozycjach (współrzędnych) na ekranie (w oknie aplikacji). Należy uwzględnić, że rozdzielczość używana przez użytkownika może być 800x600 pixeli lub 1024x768, mogą być również inne rozdzielczości. W takich warunkach napisanie aplikacji, której okno przystosowane jest do jednej konkretnej rozdzielczości spowoduje grymasy niezadowolenia wśród tych, którzy używają innych.

Dlatego w Java stworzono koncepcję kontenerów posiadających własne systemy zarządzania układem komponentów (layout managers). Komponenty "wkładane" do takich kontenerów zachowują się według ściśle określonych zasad, charakterystycznych dla danego layout managera. W pakiecie Abstract Windowing Toolkit (AWT) Javy zdefiniowano pięć klas takich obiektów:

Natomiast jednym z częściej używanych kontenerów jest Panel.

Inne, to np.

- Frame,
- Dialog
- Applet.

Stosunkowo istotną cechą każdego kontenera (także ogólniej: komponentu) są jego zalecane wymiary (preferred dimensions) raportowane poprzez metody:

- `public Dimension getPreferredSize()` - zwraca pożądane wymiary komponentu,
- `public Dimension getMinimumSize()` - zwraca najmniejsze pożądane wymiary komponentu,
- `public Dimension getMaximumSize()` - zwraca największe pożądane wymiary komponentu.

Można komponenty umieścić według następujących schematów:

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout

FlowLayout:

Prawdopodobnie najprostszym (acz napewno - nie nieprzydatnym) układem jest FlowLayout. Komponenty w tym układzie starają się ustawić w jednym wierszu, a te, które się nie zmieszczą przechodzą do następnego wiersza, gdzie sytuacja się powtarza.

Układ ten może być użyty w jednym z trzech wariantów:

FlowLayout.CENTER - środkowany (domyślny),

FlowLayout.LEFT - justowany do lewej krawędzi kontenera,

FlowLayout.RIGHT - justowany do prawej.

```
public class FlowTest
{
    public static void main(String[] args)
    {
        Frame f = new Frame("FlowTest");
        f.setLayout(new FlowLayout());
    }
}
```

```

        f.add(new Button("A"));
        f.add(new Button("B"));
        f.add(new Button("C"));
        f.add(new Button("D"));
        f.add(new Button("E"));
        f.setVisible(true);
    }
}

```

i jego efekt w oknie (Frame):

Jak widać - układ dąży do ustawienia w jednym wierszu, a kiedy to się nie udaje - w następnym. Komponenty ustawiają się kolejno z odstępami determinowanymi własnością 'hgap'. Pozostałe miejsce w rzędzie zagospodarowane jest według ustalonej opcji justowania (tutaj: domyślnej - środkowej). Następny rząd komponentów znajduje się w odległości ustalonej przez właściwość 'vgap'. Zasady ułożenia są takie same, przy czym ponieważ komponentów jest mniej - lepiej widoczna jest zasada justowania do środka ich grupy. Ustawienie opcji justowania może się odbyć przy konstrukcji obiektu menedżera układu, np: `FlowLayout flow = new FlowLayout(FlowLayout.RIGHT);` przy czym domyślny (bezparametrowy) konstruktor tworzy justowanie do środka. Metoda `setLayout()` jest używana w kontenerach do wstawienia obiektu menedżera układu i co za tym idzie - ustalenia w nim zasad ułożenia komponentów.

Preferowane rozmiary układu `FlowLayout` to:

wysokość: największa wysokość spośród grupy komponentów + wartość 'vgap',
szerokość: suma szerokości poszczególnych komponentów + 'hgap' pomnożona przez ich ilość, ułożenie możliwie wszystkich elementów w jednym szeregu.

Wielkości te zyskują szczególne znaczenie wtedy, gdy różne kontenery są "wkładane" do jednego o ustalonym układzie. Różne raportowane preferencje, co do wymiarów powodują różne (ciekawe) efekty wzajemnego "upakowania" kontenerów, a pamiętanie o tym pomoże w budowie naprawdę efektywnych interfejsów, których wygląd w różnych sytuacjach, np. zmian rozmiarów okna przez użytkownika, jest wciąż zadowalający z punktu widzenia założeń projektowych.

BorderLayout:

Rozmieszczenie brzegowe, polega na rozmieszczeniu komponentów na obrzeżach i w środku pojemnika, miejsca umieszczenia w pojemniku określane są za pomocą nazw stron świata: West, East, North, South, Center, nazywane jest także rozmieszczeniem grawitacyjnym.

NORTH		
WEST	CENTER	EAST
SOUTH		

np.

```

public void init()
{
    setLayout(new BorderLayout());
    add(BorderLayout.NORTH, new Button("Północ"));
    add(BorderLayout.SOUTH, new Button("Południe"));
    add(BorderLayout.EAST, new Button("Wschód"));
}

```

```

add(BorderLayout.WEST, new Button("Zachód"));
add(BorderLayout.CENTER, new Button("Środek"));
}

```

GridLayout

Układ ten posiada (niewidoczną) siatkę prostokątną (grid), której linie wyznaczają krawędzie, do których dosuwane są komponenty. Jeśli komponent nie mieści się wewnątrz prostokąta ("oka" siatki) zostaje częściowo ukryty.

Komponenty układane są kolejno w rzędach od lewej do prawej i od góry w dół (począwszy od lewej górnej komórki siatki).

Konstruktor GridLayout posiada dwa główne parametry: ilość wierszy i ilość kolumn, jednak tylko jeden z nich zawsze jest użyty: jeśli liczba wierszy jest różna od 0 - według niej oblicza się liczbę kolumn (na podstawie ilości komponentów). Jeśli liczba wierszy jest równa 0 - liczbę wierszy oblicza się według liczby kolumn.

np.

```
setLayout(new GridLayout(3,4));
```

przy czym jest tu popełniony pewien błąd: z powyższego wywodu wynika, że tylko jeden z głównych parametrów konstruktora jest brany pod uwagę - i tak też będzie w tym przypadku - instrukcja zostanie wykonana tak, jakby wyglądała następująco (liczba wierszy jest "ważniejsza"):

```
setLayout(new GridLayout(3,0));
```

Użycie:

```
setLayout(new GridLayout(0,0));
```

spowoduje wyjątek: `IllegalArgumentException`.

Podczas konstrukcji siatki można także podać odstęp, z jakimi komponenty mają być dosuwane do krawędzi siatki:

```
setLayout(new GridLayout(3,0,8,10));
```

Utworzy siatkę z trzema wierszami, przy czym komponenty będą układane w odległościach 8 pikseli od krawędzi pionowych (lewych) i 10 pikseli od poziomych (górnych).

np.

```

JPanel p_lewo=new JPanel();
GridLayout rozmieszczenie_siatka = new GridLayout(2,1);
// kontener p_lewo w którym komponenty umieszczone elementy będą wg siatki
p_lewo.setLayout(rozmieszczenie_siatka);

```

np.

```

Frame f = new Frame("Grid Test");
f.setLayout(new GridLayout(3,4));
for (int x = 1; x < 13; x++)
f.add(new Button(""+x));

```

Preferowane rozmiary układu GridLayout to:

wysokość: $ph = (\text{maxPrefHeight} * \text{rows}) + (\text{vgap} * (\text{rows}+1))$,

szerokość: $pw = (\text{maxPrefWidth} * \text{cols}) + (\text{hgap} * (\text{cols}+1))$,

ułożenie wszystkich możliwie z zachowaniem ich preferowanych rozmiarów.

GridBagLayout

Jest to napewno najbardziej skomplikowany układ z wszystkich dotychczas przedstawionych. Oprócz kilku cech utrudniających jego użytkowanie, ma on także parę błędów, które ujawniają się przy usunięciu lub dodaniu komponentu do układu już po jego wyświetleniu.

Układ GridBagLayout dzieli kontener na siatkę równej wielkości komórek. Jednak w przeciwieństwie do GridLayout - decyduje ile wierszy i kolumn będzie mieć siatka i pozwala komponentom zajmować więcej niż jedną jej komórkę. Aby automatyczny podział kontenera na linie siatki mógł przebiec poprawnie, należy w specjalnym obiekcie (klasy GridBagConstraints) zostawić managerowi zbiór wskazówek.

Klasa GridBagConstraints ma szereg pól:

- *gridx i gridy*: współrzędne komórki określające miejsce umieszczenia następnego komponentu; domyślna wartość to GridBagConstraints.RELATIVE, co dla gridx daje pierwszą linię z prawej strony ostatnio dodanego komponentu, a dla gridy - pierwszą poniżej jego dolnej krawędzi,
- *gridwidth i gridheight*: określają szerokość i wysokość komórek; domyślną wartością jest 1; jeśli komponent ma być ostatni w wierszu lub kolumnie - odpowiednio gridwidth lub gridheight powinno mieć wartość GridBagConstraints.REMAINDER; jeśli komponent ma być obok ostatnio dodanego w wierszu lub kolumnie - odpowiednie własności powinny mieć wartość GridBagConstraints.RELATIVE,
- *fill*: określa sposób postępowania z komponentem, który jest mniejszy od komórki siatki - GridBagConstraints.NONE (domyślna) nie pozwala zmieniać jego rozmiarów; wartości GridBagConstraints.HORIZONTAL, GridBagConstraints.VERTICAL i GridBagConstraints.BOTH zezwalają odpowiednio na poszerzenie w poziomie, pionie i w obydwu kierunkach,
- *ipadx i ipady*: określają ile pikseli jest dodawane do raportowanego rozmiaru komponentu w kierunkach x i y; piksele dodawane są po obydwu stronach komponentu, tak więc efektywny rozmiar komponentu zostanie "zwiększony" o 2*ipadx w poziomie i 2*ipady w pionie; domyślne wartości to 0,
- *insets*: obiekt klasy Insets określający ostępy między komponentem i krawędzią siatki,
- *anchor*: kiedy komponent jest mniejszy od obszaru wyświetlania; wskazuje jego pozycję w obszarze; możliwe wartości:
GridBagConstraints.CENTER (domyślna), GridBagConstraints.NORTH,
GridBagConstraints.NORTHEAST, GridBagConstraints.EAST,
GridBagConstraints.SOUTHEAST, GridBagConstraints.SOUTH,
GridBagConstraints.SOUTHWEST, GridBagConstraints.WEST i
GridBagConstraints.NORTHWEST,
- *weightx i weighty*: używane są do określenia względnych rozmiarów komponentów; mają znaczenie mnożników odpowiednich wymiarów poziomego i pionowego;

np:

```
GridBagLayout gbl = new GridBagLayout();//nowy obiekt managera
setLayout(gbl);//bieżący kontener będzie mieć ten układ
Button bt = new Button("test button");
GridBagConstraints cnstr = new GridBagConstraints();//nowy obiekt ograniczeń
(wskazówek)
cnstr.weightx=1;
cnstr.gridwidth = GridBagConstraints.RELATIVE;
cnstr.fill = GridBagConstraints.BOTH;
```

```
gbl.setConstraints(bt,cnstr);//wskazówki dotyczą przycisku bt  
add(bt);//przycisk staje się częścią bieżącego kontenera
```

BoxLayout

Wiele programistów mają kłopot z użyciem GridBagLayout

W pakiecie javax.swing.* zaoferowano prostsze rozwiązanie: BoxLayout;

Rozmieszcza komponenty poziomo lub pionowo. Przestrzeń między komponentami są kontrolowane za pomocą struts (pręty) i glue (guma lub sprężyna) i rigid (sztywna przestrzeń)

CardLayout

Układ ten jest podobny do stosu: komponenty układane są jeden na drugim - w danym momencie widoczny jest tylko jeden z nich. Taki układ może być przydatny np. przy konstrukcji interfejsów z zakładkami, lub w formie notatnika.

Dozwolone są trzy formy dla dodania managera CardLayout do kontenera:

- public void add(Component component, String key);
- public void add(String key, Component component);
- public void add(String key, Component component, int index);

Dwie pierwsze poprostu nakładają kolejną "warstwę" - czyli dokładają komponent jako ostatni. Trzecia natomiast - dodaje komponent na pozycji wskazanej przez parametr 'index'. Parametr 'key' jest unikalnym identyfikatorem używanym później do wyboru komponentu ze "stosu" CardLayout.

Do manipulacji komponentami w tym układzie służą metody:

- next(cnr);
- previous(cnr);
- show(cnr,"komponent pierwszy");

gdzie 'cnr' jest kontenerem zawierającym managera CardLayout, a łańcuch znaków "komponent pierwszy" - identyfikatorem jednego z jego komponentów.

Przykład użycia:

```
Panel p = new Panel(new CardLayout());  
p.add("one", new Button ("the first component"));  
p.add(new Button ("the second component"), "two");  
p.add("three", new Button ("the third component"));  
p.add("between two and three", new Button ("the fourth component"), 2);
```

Preferowane rozmiary układu:

wysokość: największa możliwa wysokość spośród wszystkich komponentów w układzie,

szerokość: największa możliwa szerokość spośród wszystkich komponentów w układzie.

Podsumowanie

Zostały tu przedstawione wszystkie możliwe w AWT klasy managerów układu. Każdy z nich ma swoje cechy charakterystyczne, predystynujące go do określonych typów zastosowań.

Widoczne też jest, że naprawdę dobre efekty można otrzymać zwykle przez wykorzystanie kombinacji i zagnieżdżenia kilku rodzajów układów.

Przykład (nie oceniany)

Pole tekstowe z etykietą w oknie:

```
Panel p = new Panel(new BorderLayout());  
Label nameLabel = new Label("Name:");  
TextField entry = new TextField();  
p.add(nameLabel, BorderLayout.WEST);
```

```
p.add(entry, BorderLayout.CENTER);
Panel p2 = new Panel(new BorderLayout());
p2.add(p, BorderLayout.NORTH);
```

i jego wygląd w różnych sytuacjach (zmian rozmiarów okna przez użytkownika):

A oto jak małe niedopatrzenie może zaskoczyć projektanta:

```
Panel p = new Panel(new BorderLayout());
Label nameLabel = new Label("Name:");
TextField entry = new TextField();
p.add(nameLabel, BorderLayout.WEST);
p.add(entry, BorderLayout.CENTER);
```

Została tu także wspomniana klasa Insets. Każdy kontener posiada obiekt tej klasy, który służy do określenia "marginesów", z jakimi rysowane są krawędzie komponentu. Obiekt taki jest zwracany przez metodę `getInsets()`, dziedziczoną z klasy `Container`. Domyślna metoda zwraca obiekt wskazujący na zerowe wartości marginesów górnego, lewego, dolnego i prawego (w tej kolejności). Aby to zmienić wystarczy przykryć tę metodę we własnej klasie nową, np.:

```
public Insets getInsets() {return (new Insets(10,5,10,5));}
```

Przykład (nie oceniany)

Jest to klasa rozszerzająca `Panel` tak, aby jego wygląd przypominał trójwymiarową płytkę (jak `TPanel` w C++Builderze, czy Delphi);

```
class BorderPanel extends Panel
{
    private static final Insets insets = new Insets(10,10,10,10);
    public Insets getInsets()
    {return insets;}
    public void paint(Graphics g)
    {
        Dimension size = getSize();
        g.setColor(getBackground());
        g.draw3DRect(5,5,size.width-11, size.height-11, true);
    }
}
```

Inne własności panelu

`pan.setPreferredSize(roz)` → ustala rozmiar panelu

`pan.setBackground(kolor)` → tło

`pan.add(składnik)` → dodaje do panelu określony składnik (np. przycisk lub pole tekstowe)

Przykład (nie oceniany)

Rozszerzenie poprzedniego przykładu w celu prostego sformatowania okna.

```

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

class panel
{
    public static void main(String[] args)
    {
        JFrame okno=new JFrame();
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setVisible(true);
        okno.setResizable(false);
        okno.setSize(400,200);
        okno.setLocation(200,200);
        okno.setTitle("glowne okno programu");
        Color c=new Color(250,0,0);                //definiujemy kolor tła panelu
        Dimension rozmiar=new Dimension(370,370); //definiujemy rozmiary panelu.
        //tworzymy panel w glownym oknie programu
        JPanel p=new JPanel();
        p.setLayout(new FlowLayout());
        p.setPreferredSize(rozmiar);
        p.setBackground(c);
        okno.setContentPane(p);                    //ładujemy stworzony panel do obiektu okno.
        okno.pack();                               //metoda pack powoduje wyświetlenie panelu w oknie.





















    }
}

```

ZADANIE 25

Zmień zadanie/przykład poprzedni tak, aby:

- JFrame okno_nr_z_dziennika=new JFrame();
- w tytule okna wyświetliło się Twoje nazwisko
- można było skalować myszką okno (sprawdź czy można zmienić wielkość okna myszką),
- zmień kolor tła na jeden z palety kolorów przedstawionych poniżej. Kolory są umieszczone poniżej wg alfabetu . Pierwszy kolor *aliceblue* ma numer 1 i kolejne wg alfabetu mają numery kolejne (czyli numerujemy wierszami). Twój numer to 75+numer z dziennika. Zmiana kolorów z użyciem Color(R,G,B), będziesz musiał przeliczyć system szesnastkowy na trzy liczby systemu dziesiętnego (R, G, B). 1-2 znak w szesnastkowym to R, 3-4 znak w szesnastkowym to G, 5-6 znak w szesnastkowym to B
- rozmiar okna (400+nr_z_dziennika, 200+nr_z_dziennika) z użyciem Dimension
- umieszczenie okna (200+nr_z_dziennika, 200+nr_z_dziennika)

	aliceblue #F0F8FF		Antiquewhite #FAEBD7		aqua #00FFFF		aquamarine #7FFFD4
	Aurze #F0FFFF		Beige #F5F5DC		bisque #FFE4C4		black #000000
	blanchedalmond #FFEBCD		Blue #0000FF		blueviolet #8A2BE2		brown #A52A2A
	burlywood #DEB887		Cadetblue #5F9EA0		chartreuse #7FFF00		chocolate #D2691E
	Coral #FF7F50		Cornflowerblue #6495ED		cornsilk #FFF8DC		crimson #DC143C

 Cyan #00FFFF	 Darkblue #00008B	 darkcyan #008B8B	 darkgoldenrod #B8860B
 darkgray #A9A9A9	 Darkgreen #006400	 darkkhaki #BDB76B	 darkmagenta #8B008B
 darkolivegreen #556B2F	 Darkorange #FF8C00	 darkorchid #9932CC	 darkred #8B0000
 darksalomon #E9967A	 Darkseagreen #8FBC8F	 darkslateblue #483D8B	 darkslategray #2F4F4F
 darkturquoise #00CED1	 Darkviolet #9400D3	 deeppink #FF1493	 deepskyblue #00BFFF
 Dimgray #696969	 Dodgerblue #1E90FF	 firebrick #B22222	 floralwhite #FFFAF0
 forestgreen #228D22	 Fuchsia #FF00FF	 ghostwhite #F8F8FF	 gainsboro #DCDCDC
 Gold #FFD700	 Goldenrod #DAA520	 gray #808080	 green #008000
 greenyellow #ADFF2F	 Honeydew #F0FFF0	 hotpink #FF69B4	 indianred #CD5C5C
 Indigo #4B0082	 Ivory #FFFFFF0	 khaki #F0E68C	 lavender #E6E6FA
 lavenderblush #FFF0F5	 Lawngreen #7CFC00	 lemonchiffon #FFFACD	 lightblue #ADD8E6
 lightcoral #F08080	 Lightcyan #E0FFFF	 lightgoldenrodyellow #FAFAD2	 lightgreen #90EE90
 lightgrey #D3D3D3	 Lightpink #FFB6C1	 lightsalmon #FFA07A	 lightseagreen #20B2AA
 lightskyblue #87CEFA	 Lightslategray #778899	 lightsteelblue #B0C4DE	 lightyellow #FFFFE0
 Lime #00FF00	 Limegreen #32CD32	 linen #FAF0E6	 magenta #FF00FF
 Maroon #800000	 mediumaquamarine #66CDAA	 mediumblue #0000CD	 mediumorchid #BA55D3
 mediumpurple #9370DB	 mediumseagreen #3CB371	 mediumslateblue #7B68EE	 mediumspringgreen #00FA9A
 mediumturquoise #48D1CC	 mediumvioletred #C71585	 midnightblue #191970	 mintcream #F5FFFA
 Mistrose #FFE4E1	 Moccasin #FFE4B5	 navajowhite #FFDEAD	 navy #000080
 Oldlace #FDF5E6	 Olive #808000	 olivedrab #6B8E23	 orange #FFA500
 Orangered #FF4500	 Orchid #DA70D6	 palegoldenrod #EEE8AA	 palegreen #98FB98
 Paleturquoise #AFEEEE	 Palevioletred #DB7093	 papayawhip #FFEDF5	 peachpuff #FFDAB9
 Peru #CD853F	 Pink #FFC0CB	 plum #DDA0DD	 powderblue
 Purple #800080	 Red #FF0000	 rosybrown #BC8F8F	 royalblue #4169E1
 Saddlebrown #8B4513	 Salomon #FA8072	 sandybrown #F4A460	 seagreen #2E8B57
 Seashell #FFF5EE	 Sienna #A0522D	 silver #C0C0C0	 skyblue #87CEEB
 Slateblue #6A5ACD	 Slategrey #708090	 snow #FFFAFA	 springgreen #00FF7F
 Steelblue #4682B4	 Tan #D2B48C	 teal #008080	 thistle #D8BFD8
 Tomato #FF6347	 Turquoise #40E0D0	 violet #EE82EE	 wheat #F5DEB3
 White #FFFFFF	 Whitesmoke #F5F5F5	 yellow #FFFF00	 yellowgreen #9ACD32

Komponenty

Każda aplikacja okienkowa, może mieć takie elementy jak jak:

- menu (zwyczajne, kaskadowe)
- przyciski,
- etykiety,
- pola tekstowe
- listy rozwijane

Pakiet javax.swing zawiera oczywiście odpowiednią porcję klas, które pozwalają na zastosowanie tego rodzaju komponentów.

Praktycznie wszystkie graficzne komponenty, które pozwalają na zastosowanie typowych elementów środowiska okienkowego, dziedziczą, pośrednio lub bezpośrednio, z klasy JComponent, a zatem są wyposażone w jej metody. Metod tych jest dosyć dużo, ich pełną listę można znaleźć w dokumentacji JDK. W tabeli zostały natomiast zebrane niektóre z nich, przydatne przy wykonywaniu typowych operacji. Większość została odziedziczona z klasy Component biblioteki AWT.

Typ zwracany	Metoda	Opis
Void	addKeyListener(KeyListener l)	Ustala obiekt, który będzie obsługiwał zdarzenia związane z klawiaturą.
Void	addMouseListener(MouseListener l)	Ustala obiekt, który będzie obsługiwał zdarzenia związane z klikaniem przyciskami myszy
Void	addMouseMotionListener(MouseMotionListener l)	Ustala obiekt, który będzie obsługiwał zdarzenia związane z ruchami myszy
Boolean	contains(int x, int y)	Sprawdza, czy komponent zawiera punkt współrzędnych x, y.
Boolean	contains(Point p)	Sprawdza, czy komponent zawiera punkt wskazywany przez argument p.
Color	getBackground()	Pobiera kolor tła komponentu
Rectangle	getBounds()	Zwraca rozmiar komponentu w postaci obiektu klasy Rectangle
Cursor	getCursor()	Zwraca kursor związany z komponentem
Font	getFont()	Zwraca czcionkę związaną z komponentem
FontMetrics	getFontMetrics(Font font)	Zwraca obiekt opisujący właściwości czcionki związanej z komponentem
Color	getForeground()	Zwraca pierwszoplanowy kolor komponentu.
Graphics	getGraphics()	Zwraca tzw. graficzny kontekst urządzenia, obiekt pozwalający na

		wykonywanie operacji graficznych na komponencie.
Int	getHeight()	Zwraca wysokość komponentu
String	getName()	Zwraca nazwę komponentu
Container	getParent()	Zwraca obiekt nadrzędny komponentu
Dimension	getSize()	Zwraca rozmiary komponentu w postaci obiektu klasy Dimension
Int	getWidth()	Zwraca szerokość komponentu
Int	getX()	Zwraca współrzędną x położenia komponentu
Int	getY()	Zwraca współrzędną y położenia komponentu
Void	Reprint()	Odrysowuje cały obszar komponentu.
Void	Repaint(int x, int y,int width, int height)	Odrysowuje wskazany obszar komponentu
Void	setBackground(Color c)	Ustala kolor tła komponentu
Void	setBounds(int x, int y,int width, int height)	Ustala rozmiar i położenie komponentu. 1.1
Void	setBounds(Rectangle r)	Ustala rozmiar i położenie komponentu
Void	setCursor(Cursor cursor)	Ustawia rodzaj kursora przypisany komponentowi
Void	setFont(Font f)	Ustawia czcionkę przypisaną komponentowi
Void	setLocation(int x, int y)	Zmienia położenie komponentu
Void	setLocation(Point p)	Zmienia położenie komponentu
Void	setName(String name)	Ustala nazwę komponentu
Void	setSize(Dimension d)	Ustala rozmiary komponentu.
Void	setSize(int width, int height)	Ustala rozmiary komponentu
Void	setVisible(boolean b)	Wyświetla lub ukrywa komponent

Pola tekstowe

Pola tekstowe występują w kilku rodzajach:

- TextField, → pozwala na wprowadzenie tekstu w jednej linii
- JPasswordField,
- JFormattedTextField,
- JTextArea, → tekstu wielowierszowego.
- JEditorPane
- JTextPane.

Wszystkie te klasy dziedziczą bezpośrednio bądź pośrednio z JTextComponent, która z kolei dziedziczy z JComponent.

Pola tekstowe jednolinijkowe- klasa JTextField

Klasa JTextField tworzy jednowierszowe pole tekstowe.

Oferuje nam ona pięć konstruktorów.

Konstruktor	Opis
-------------	------

<code>TextField()</code>	Tworzy nowe, puste pole tekstowe
<code>TextField(Document doc, String text, int columns)</code>	Tworzy nowe pole tekstowe o zadanej liczbie kolumni zawartości, zgodne z modelem dokumentu wskazanym przez argument doc.
<code>TextField(int columns)</code>	Tworzy nowe pole tekstowe o zadanej liczbie kolumn(znaków).
<code>TextField(String test)</code>	Tworzy nowe pole tekstowe zawierające wskazany tekst
<code>TextField(String text, int columns)</code>	Tworzy nowe pole tekstowe o zadanej liczbie kolumnzawierające wskazany tekst.

Uwagi:

- W polu tekstowym można wypisywać jakieś informacje (Output) lub te informacje pobierać (Input).
- Pole tekstowe jest obiektem o określonej nazwie.

Pole tekstowe z określoną ilością znaków.

`TextField nazwa_pola=new TextField(int ilosc_znakow);`

Pole tekstowe z napisem (wówczas jego rozmiar jest ustalany na podstawie ilości znaków w napisie):

`TextField nazwa_pola=new TextField("napis inicjujacy");`

Pobiera tekst z pola tekstowego

**`TextField pole=new TextField(s);`
`s=pole.getText()`**

Ustawia parametry czcionki użytej w polu tekstowym

`pole.setFont(czcionka)`

Wstawia napis do pola tekstowego

`pole.setText(s)`

Możliwość wpisywania tekstu z zewnątrz do pola tekstowego.

`pole.setEditable(war)`

jeśli war=false to do pola tekstowego nie można wpisywać tekstu z zewnątrz (brak interaktywności)

Czcionki w Java

Klasa Font czcionkę reprezentuje obiekt klasy Font tworzony za pomocą konstruktora:
`Font(krój, styl, rozmiar)`

Styl :

Font.PLAIN - czcionka zwykła,
Font.BOLD – czcionka pogrubiona,
Font.ITALIC – czcionka pochylona.

Krój :

w środowisku Java 2 jest dostępne tylko 5 czcionek logicznych

Serif,
SansSerif,
Monospaced,
Dialog,
DialogInput.

Domyślnie, jeżeli nie ustawimy żadnej czcionki to przypisywana jest czcionka Dialog.

Rozmiar :

Przykład (nie jest oceniane)

- Treść zadania:
Tworzone dwóch pól tekstowych. Czcionka w jednym z nich jest standardowa a w drugim parametry czcionki są zmieniane.

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

class Przyklad102
{
    public static void main(String[] args)
    {
        JFrame okno=new JFrame();
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setVisible(true);
        JPanel p=new JPanel();
        p.setLayout(new FlowLayout());
        p.setPreferredSize(new Dimension(200,200));
        p.setBackground(Color.blue);
        String s1="pole 1";
        String s2="pole 2";
        Font czcionka=new Font("Dialog", Font.ITALIC+Font.BOLD,20);
        JTextField pole1=new JTextField(s1);
        JTextField pole2=new JTextField(s2);

        pole1.setFont(czcionka); //tworzymy 2 pola tekstowe i inicjujemy je.
        //ustalamy parametry czcionki wyświetlanej w polu tekstowym 1
        p.add(pole1); //dodajemy utworzone pola tekstowe do panelu.
        p.add(pole2);
        okno.setContentPane(p);
        okno.pack();
    }
}
```

ZADANIE 26

Zadanie praktyczne.

Zmień zadanie/przykład poprzedni tak, aby:

- JFrame okno_nr_z_dziennika=new JFrame();
- w tytule okna wyświetliło się Twoje nazwisko
- można było skalować myszką okno (sprawdź czy można zmienić wielkość okna myszką),

- zmień kolor tła na jeden z palety kolorów przedstawionych powyżej. Kolory są umieszczone poniżej wg alfabetu . Pierwszy kolor alicebblue ma numer 1 i kolejne wg alfabetu mają numery kolejne. Twój numer to 27+numer z dziennika. Zmiana kolorów z użyciem Color(R,G,B)
- rozmiar okna (300+nr_z_dziennika, 250+nr_z_dziennika) z użyciem Dimension
- umieszczenie okna (250+nr_z_dziennika, 250+nr_z_dziennika)
- trzy pola tekstowe o różnych wielkościach z innymi rodzajami czcionek. Umieszczone jedno pod drugim. Napisy to:
 - pierwsze pole →nazwisko oraz nazwa koloru tła (kolor tła→Twój numer to 17+numer z dziennika) np. Steelblue
 - drugie pole →imię oraz nazwa koloru tła (kolor tła→Twój numer to 37+numer z dziennika)np. Tan
 - trzecie pole →telefon (fałszywy) oraz nazwa koloru tła (kolor tła→Twój numer to 47+numer z dziennika) np. Sienna

Uzyskanie koloru tła dla pierwszego, drugiego, trzeciego pola wykonujemy następująco: najpierw definiujemy trzy obiekty kolorów np. C1, C2, C2 [z wykorzystaniem RGB] następnie uruchamiamy metodę
Np. pole1.setBackground (C1);

Pole tekstowe rozmieszczone wg GridLayout ([kliknij aby przeczytać](#)), zastanów się ile będzie wierszy a ile kolumn. Czyli musisz zmienić rozmieszczenie z przykładu p.setLayout(new FlowLayout()); na GridLayout.

Koniec zadania 26.

Etykieta *JLabel*

Etykiety służą do wyświetlania statycznego tekstu. Aby utworzyć etykietę, należy skorzystać z klasy *JLabel*. Komponent *JLabel* (w odróżnieniu od protoplasty – komponentu java.awt.Label) ma ponadto możliwość wyświetlania obrazków, elementów HTML oraz obramowania.

Metody klasy *JLabel*:

- **JLabel()** – konstruktor domyślny – tworzy pustą etykietę,
 - **JLabel(String text)** – tworzy etykietę z napisem text,
 - **JLabel(Icon image)** – tworzy etykietę z obrazkiem (ikonką) image,
 - **JLabel(String text, Icon icon, int hAlignment)** – tworzy etykietę z tekstem i grafiką, ostatni parametraz oznacza wyrównywanie (np. SwingConstants.LEFT dla wyrównania do lewej),
 - void setIcon(**Icon icon**) – ustawia ikonkę na etykiecie na podany obraz,
 - void setText(String text) – ustawia tekst na etykiecie na podany jako parametr,
- Jeżeli w treści etykiety umieścimy tekst zaczynający się od "<html>", wówczas zostanie on sformatowany jako HTML.

Po utworzeniu etykiety znajdujący się na niej tekst można pobrać za pomocą metody getText, jeśli natomiast chcemy go zmienić, korzystamy z metody setText. Etykietę umieszczamy w oknie lub na innym komponencie, wywołując metodę add.

Przykład 22

Treść przykładu

Wyświetlanie tekstu w Label o określonym kolorze w okienku.

Zadanie praktyczne:

- Klasa publiczna to Twoje nazwisko_p22
- Wpisz i uruchom przykład.

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

public class AplikacjaOkienkowa
{
    public static void main(String args[])
    {
        JFrame okno=new JFrame("Moja aplikacja.");
        okno.setSize(200,200);
        JPanel panel=new JPanel();
        panel.setBackground(new Color(210,246,255));
        JLabel etykieta=new JLabel("Java jest wspaniała!"); // nowa etykieta z napisem
        etykieta.setForeground(Color.red); // określenie koloru napisu
        panel.add(etykieta); // dodanie etykiety do panelu
        okno.getContentPane().add(panel); // dodanie panelu do powierzchni okna
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setLocationRelativeTo(null);
        okno.setVisible(true);
    }
}
```

Przykład 23

Inny sposób użycia Label → opis patrz poniżej

Zadanie praktyczne:

- Klasa publiczna to Twoje nazwisko_p23
- Wpisz i uruchom przykład.

```
import javax.swing.*;

public
class Aplikacja extends JFrame
{
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        JLabel label1 = new JLabel("Pierwsza etykieta");
        label1.setBounds(100, 40, 120, 20);
        JLabel label2 = new JLabel();
        label2.setText("Druga etykieta");
        label2.setBounds(100, 60, 120, 20);
        add(label1);
        add(label2);
        setSize( 350, 200);
        setVisible(true);
    }
}
```

```

public static void main(String args[])
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new Aplikacja();
        }
    });
}
}

```

Opis

Aplikacja wykorzystuje oba wymienione wyżej sposoby tworzenia etykiet. W pierwszym przypadku (obiekt label1) już w konstruktorze przekazywany jest tekst, jaki ma być wyświetlany, w przypadku drugim (obiekt label2) tworzona etykieta jest pusta, a tekst jest jej przypisywany za pomocą metody setText. Rozmiary oraz położenie etykiet są ustalane dzięki metodzie setBounds. Pierwsze dwa argumenty tej metody określają współrzędne x i y, natomiast dwa kolejne szerokość oraz wysokość. Etykiety są dodawane do okna aplikacji za pomocą instrukcji:

```
add(label1);
```

```
add(label2);
```

Metoda setLayout ustala sposób rozkładu elementów w oknie. Przekazany jej argument null oznacza, że rezygnujemy z automatycznego rozmieszczania elementów i będziemy je pozycjonować ręcznie. W tym przypadku oznacza to, że argumenty metody setBounds będą traktowane jako bezwzględne współrzędne okna aplikacji.

ZADANIE 27

Na podstawie przykładu poprzedniego wykonaj następujące zadanie:

- class Okienko_nazwisko_ucznia_z24
- Wykonaj wizytówkę z użyciem **czterech label** z obramowaniem z gwiazdek w postaci:

```

*****
*                      Nazwisko                      *
*                      klasa                          *
*****

```

- Każdy wiersz innym kolorem.

Do zmiany kolorów użyj np. `label1.setForeground(kolor1);`

Gdzie **kolor1** jest zdefiniowanym nowym obiektem kolor RGB np.

```
Color kolor1=new Color(250,0,0);
```

Koniec zadania 27.

Przykład 24

Zadanie praktyczne:

- Klasa publiczna to Twoje nazwisko_p24
- Wpisz przykład i uruchom go.

Aby program działał poprawnie konieczny będzie plik Skype.png weź go od nauczyciela lub z Internetu.

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

public class AplikacjaOkienkowa
{
    public static void main(String args[])
    {
        Okno okno=new Okno();
    }
}

class Okno extends JFrame //definicja klasy Okno jako pochodną klasy JFrame
{
    public Okno()          // utworzenie bezparametrowego konstruktora
    {
        setSize(400,200);
        setTitle("Skype");
        Panel panel=new Panel(); //utworzenie nowego obiektu
                                // panel
        Border krawedz=BorderFactory.createLineBorder(Color.red,5);
                                //utorzenie obiektu krawedz
                                //czerwona o grubości 5
        panel.setBorder(krawedz);
                                //ustawienie krawędzi obiektu panel
        add(panel);
                                // dodanie obiektu panel do okna
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);
    }
}

class Panel extends JPanel //definicja klasy Panel jako pochodną klasy JPanel
{
    public void paintComponent(Graphics g)
        // do rysowania własnego panelu służy metoda
    {
        // paintComponent() z parametrem g typu Graphics
        ImageIcon im=new ImageIcon("Skype.png");
        // nowy obiekt im klasy ImageIcon z konstruktorem
        // ImageIcon gdzie podajemy nazwę pliku graficznego
        // może być ze ścieżką dostępu
        Image img=im.getImage();
        // utworzenie obiektu img klasy Image
        // o wartościach obiektu im pobranie wartości z
        // użyciem metody getImage()
        g.drawImage(img,10,10,null,null);
        // rysowanie obiektu img, 10,10 oznacza miejsce rysowania
    }
}
```

```

        // pierwszy null oznacza przezroczystość tła obrazka
        // drugi null ??? coś z ImageObserver
g.setColor(Color.red);
        // wstawienie koloru czerwonego
Font czcionka=new Font("Monospaced ",Font.BOLD,30);
        // zdefiniowanie obiektu czcionka o ustalonych parametrach
g.setFont(czcionka);
        // wstawienie czcionki
g.drawString("Skype",200,50);
        // wykonanie napisu Skype w miejscu (200,50)
g.setColor(Color.green);
g.fillRect (220,100,100,57);
        //rysowanie wypełnionego prostokąta zielonym kolorem
    }
}

```

ZADANIE 25

Napisz program wyświetlający, co najmniej cztery pliki graficzne z opisami odpowiadającymi grafice. Opis umieść pod grafiką tak jak wykonuje się teksty. Całość ma tworzyć śmieszną przyzwoitą historyjkę. Pliki pobierz z Internetu. Opisy wymyśl sam.

- Klasa zawierająca main ma mieć nazwę Nazwisko_ucznia_z25 (bez polskich liter).
- Klasa pochodną klasy JFrame ma mieć nazwę Okno_nazwisko_ucznia.
- Wielkość okna 600+nr_z_dziennika, 400+nr_z_dziennika
- Tytuł okna nazwisko ucznia.
- Klasa pochodną klasy JPanel ma mieć nazwę Pan_nazwisko_ucznia.
- Ramka grubość numer_z_dziennika, kolor zielony

Koniec zadania 25

Grafika 2D i 3D

Klasa Graphics

umożliwia również rysowanie prostych figur:

drawString(String s, int x, int y)→ wyświetla tekst s począwszy od punktu(x,y)

drawLine(int x1,int y2, int x2, int y2)→ rysowanie linii

drawOval (int x, int y , int width, int height)→ rysowanie okręgu(elipsy) począwszy od punktu(x,y) o szerokości i wysokości, gdy szerokość=wysokość to okrąg.

fillOval (int x, int y , int width, int height)→ rysowanie wypełnionego koła (elipsy)

drawRect (int x, int y , int width, int height) →rysowanie Prostokąta począwszy od punktu(x,y) o szerokości i wysokości, gdy szerokość=wysokość to kwadrat

fillRect (int x, int y , int width,int height) →rysowanie wypełnienia Prostokąta

drawPolygon(Polygon p)→ rysowanie wielokąta

fillPolygon(Polygon p)→ rysowanie wypełnionego wielokąta

//parametr p jest typu Polygon, co oznacza, że musimy stworzyć obiekt takiego typu.

Można to dokonać za pomocą konstruktora np.:

Polygon (int xPoints Points[],int nPoints[]) [],int y)

Tablice xPoints i yPoints muszą zawierać współrzędne kolejnych punktów wielokąta.

Parametr nPoints oznacz liczbę tych punktów.

Deklaracja metody	Opis
<code>void drawLine(int x1, int y1, int x2, int y2)</code>	Rysuje linię rozpoczynającą się w punkcie o współrzędnych x1, y1 i kończącą się w punkcie x2, y2.
<code>void drawOval(int x, int y, int width, int height)</code>	Rysuje owal wpisany w prostokąt opisany parametrami x, y oraz width i height.
<code>void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)</code>	Rysuje wielokąt o punktach wskazywanych przez tablice xPoints i yPoints. Liczbę segmentów linii, z których składa się figura, wskazuje parametr nPoints.
<code>void drawPolygon(Polygon p)</code>	Rysuje wielokąt opisany przez argument p.
<code>void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)</code>	Rysuje sekwencję połączonych ze sobą linii o współrzędnych zapisanych w tablicach xPoints i yPoints. Liczba segmentów jest określona przez argument nPoint.
<code>void drawRect(int x, int y, int width, int height)</code>	Rysuje prostokąt zaczynający się w punkcie o współrzędnych x, y oraz szerokości i wysokości określonej przez argumenty width i height.
<code>void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	Rysuje prostokąt o zaokrąglonych rogach zaczynający się w punkcie o współrzędnych x, y oraz szerokości i wysokości określonej przez argumenty width i height. Stopień zaokrąglenia rogów jest określany przez argumenty arcWidth i arcHeight.
<code>void fillOval(int x, int y, int width, int height)</code>	Rysuje koło lub elipsę wpisaną w prostokąt opisany parametrami x, y oraz width i height.
<code>void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)</code>	Rysuje wypełniony bieżącym kolorem wielokąt o punktach wskazywanych przez tablice xPoints i yPoints. Liczbę segmentów linii, z których składa się figura, wskazuje argument nPoints.
<code>void fillPolygon(Polygon p)</code>	Rysuje wypełniony bieżącym kolorem wielokąt opisany przez argument p.
<code>void fillRect(int x, int y, int width, int height)</code>	Rysuje wypełniony bieżącym kolorem prostokąt zaczynający się w punkcie o współrzędnych x, y oraz szerokości i wysokości określonej przez argumenty width i height.
<code>void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	Rysuje wypełniony bieżącym kolorem prostokąt o zaokrąglonych rogach, zaczynający się w punkcie o współrzędnych x, y oraz szerokości i wysokości określonej przez argumenty width i height. Stopień zaokrąglenia rogów jest określany przez argumenty arcWidth i arcHeight.

Przykład 25

Zadanie praktyczne:

- Klasa publiczna to Twoje nazwisko_p25
- Wpisz przykład i uruchom go.

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.border.*;

public class AplikacjaOkienkowa
{
    public static void main(String args[])
    {
        Okno okno=new Okno();
    }
}

class Okno extends JFrame
{
    public Okno()
    {
        setSize(400,200);
        setTitle("Grafika 2D");
        Panel panel=new Panel();
        Border krawedz=BorderFactory.createLineBorder(Color.red,5);
        panel.setBorder(krawedz);
        add(panel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);
    }
}

class Panel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2=(Graphics2D)g;
        //rzutowanie obiektu klasy Graphics na obiekt
        // jej klasy pochodnej Graphics2D, zdefiniowanie obiektu g2
        // przypisujemy obiekt g
        g2.setColor(Color.black);
        Rectangle2D prost=new Rectangle2D.Float(5,5,380,155);
        // dwie pierwsze to współrzędne lewego górnego rogu
        // trzecia i czwarta to szerokość i wysokość
        //nowy obiekt prost rysujący prostokąt
        g2.fill(prost);
        // wypełnienie prostokąta
        g2.setColor(Color.green);
        Ellipse2D elipsa=new Ellipse2D.Float(5,5,380,155);
```



```
//nowy obiekt elipsa rysujący elipsę
// dwie pierwsze to współrzędne lewego górnego rogu
// trzecia i czwarta to szerokość i wysokość
// prostokąta gdzie zmieści się elipsa
g2.fill(elipsa);
g2.setColor(Color.white);
g2.draw(elipsa);
g2.setColor(Color.yellow);
g2.fill3DRect(50,50,100,50,true);
// rysowanie prostokata w 3D
// true-wrażenie uniesienia prostokąta
}
}
```

ZADANIE 28

Czynności praktyczne:

- Klasa zawierająca main ma mieć nazwę Nazwisko_ucznia_z28 (bez polskich liter).
- Klasa pochodną klasy JFrame ma mieć nazwę Okno_nazwisko_ucznia.
- Wielkość okna 500+nr_z_dziennika, 300+nr_z_dziennika
- Tytuł okna nazwisko ucznia.
- Klasa pochodną klasy JPanel ma mieć nazwę Pan_nazwisko_ucznia.
- Ramka grubość numer_z_dziennika, kolor czerwony
- Na podstawie przykładu powyżej narysuj bryłę: ścięty stożek (dla numerów nieparzystych w dzienniku) leżący walec (dla numerów parzystych w dzienniku). Figurę będącą podstawą zapełnij, figury te wykonaj jako elipsa, zaznacz promień i wysokość wewnątrz figury, zapisz obok bryły wzory na pole powierzchni całkowitej oraz objętość.

Koniec zadania 28

Zadanie

Napisz program wyświetlający, . ZMIENIĆ

- Klasa zawierająca main ma mieć nazwę Nazwisko_ucznia (bez polskich liter).
- Klasa pochodną klasy JFrame ma mieć nazwę Okno_nazwisko_ucznia.
- Wielkość okna 600+nr_z_dziennika, 400+nr_z_dziennika
- Tytuł okna nazwisko ucznia.
- Klasa pochodną klasy JPanel ma mieć nazwę Pan_nazwisko_ucznia.
- Ramka grubość numer_z_dziennika, kolor zielony

Pola tekstowe wielolinijkowe- klasa JTextArea

Można utworzyć także pola tekstowe, w których tekst wyświetlany jest w więcej niż jednej linijce. Takie pola tekstowe są obiektami klasy JTextArea.

Definicje takich obiektów mogą wyglądać następująco:

int x,y; //x-ilość kolumn, y - ilość wierszy

```
String s="jakis napis";
```

```
    JTextArea pole=new JTextArea(x,y);  
lub  
    JTextArea pole=new JTextArea(s,x,y);
```

Metody dostępne w klasie JTextArea:

metoda opis

String s;

int i,pos,rozmiar;

Font czcionka;

JTextArea pole;

```
pole.append(s)  
dopisuje napis s do końca pola tekstowego
```

```
i=pole.getLineCount()  
podaje ilość wierszy w polu tekstowym
```

```
pole.setLineWrap(true)  
zapobiega rozszerzaniu wierszy (łamię wiersze)
```

```
pole.setWrapStyleWord(true)  
jeśli wyrazy nie mieszczą się na końcu linii są wówczas przenoszone do nowej linii
```

```
pole.insert(s,pos)  
wstawia napis s na pozycji pos
```

```
pole.setFont(czcionka)  
ustala parametry czcionki
```

```
pole.setTabSize(rozmiar)  
ustawia rozmiar tabulacji
```

```
pole.setCaretPosition(rozmiar)  
ustala pozycje kursora
```

```
s=pole.getText()  
pobiera tekst z całego pola tekstowego
```

```
i=pole.getLineStartOffset(n_line)  
podaje numer znaku stojącego na początku linii n_line
```

```
i=pole.getLineEndOffset(n_line)  
podaje numer znaku stojącego na końcu linii n_line
```

Przykład 27

- Treść zadania:
Tworzenie pola tekstowego wielolinijkowego.
- Nazwa klasy to nazwisko_ucznia_p27

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

class Tekst_2
{
    public static void main(String[] args)
    {
        JFrame okno=new JFrame();
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setVisible(true);
        JPanel p=new JPanel();
        p.setLayout(new FlowLayout());
        p.setPreferredSize(new Dimension(200,200));
        p.setBackground(Color.blue);
        String s="pole tekstowe";
        Font czcionka=new Font("Dialog", Font.ITALIC+Font.BOLD,10);
        JTextArea pole=new JTextArea(s,10,10);
        //tworzymy pole tekstowe o szerokości 10 znaków i wysokości równej 10.
        pole.setEditable(true);
        pole.setFont(czcionka);
        pole.setLineWrap(true);

        //widoczny pasek przewijania w polu
        pole.setWrapStyleWord(true);
        //do następnego wiersza można przenosić nie tylko jeden znak,
        //lecz także całe słowo.
        pole.setTabSize(5);
        //ustawia rozmiar tabulacji dla setCaretPosition(rozmiar)
        pole.setCaretPosition(1);
        //ustala początkową pozycję kursora tekstowego w polu tekstowym
        pole.setBackground(Color.black);
        //ustalamy kolor tła pola tekstowego
        pole.setForeground(Color.white);
        //ustalamy kolor czcionki.

        p.add(pole);
        okno.setContentPane(p);
        okno.pack();
    }
}
```

Klasa JTextArea pozwala na tworzenie komponentów umożliwiających wprowadzenie większej ilości tekstu.

Konstruktor	Opis
JTextArea()	Tworzy pusty komponent JTextArea.
JTextArea(Document doc)	Tworzy nowe pole tekstowe zgodne z

	modelem dokumentu wskazanym przez argument doc.
JTextArea(Document doc, String text, int rows, int columns)	Tworzy nowe pole tekstowe zgodne z modelem dokumentu wskazanym przez argument doc, o zadanej zawartości oraz liczbie wierszy i kolumn.
JTextArea(int rows, int columns)	Tworzy pusty komponent JTextArea o określonej liczbie wierszy i kolumn.
JTextArea(String text)	Tworzy komponent JTextArea zawierający tekst określony przez argument text
JTextArea(String text, int rows, int columns)	Tworzy komponent JTextArea zawierający tekst określony przez argument text, o liczbie wierszy i kolumn określonej argumentami rows i columns.

ZADANIE 29

Czynności praktyczne:

- Klasa zawierająca main ma mieć nazwę Nazwisko_ucznia_z27 (bez polskich liter).
- Dwa pola tekstowe
- Wymiar pola tekstowego pierwszego: ilość wierszy $5 + (\text{reszta z dzielenia numer_z_dziennika przez } 3)$, ilość kolumn $11 + (\text{reszta z dzielenia numer_z_dziennika przez } 2)$,
- Wymiar pola tekstowego drugi: ilość wierszy $7 + (\text{reszta z dzielenia numer_z_dziennika przez } 2)$, ilość kolumn $13 + (\text{reszta z dzielenia numer_z_dziennika przez } 3)$,
- Tekst pierwszego pola to: numer w dzienniku oraz nazwisko ucznia,
- Tekst drugiego pola to: nazwa oraz skład (nazwisko+instrument) ulubionego zespołu muzycznego,
- Kolor tła panelu nr_z_dzienika +38,
- Rozkład elementów FlowLayout()
- Wielkość okna panelu szerokość= $400 + \text{nr_z_dziennika}$, wysokość= $260 + \text{nr_z_dziennika}$
- Czcionka w pierwszym oknie: wielkość $10 + (\text{reszta z dzielenia numer_z_dziennika przez } 2)$, pogrubiona, Serif, kolor czcionki nr_z_dzienika +8,
- Czcionka w drugim oknie: wielkość $10 + (\text{reszta z dzielenia numer_z_dziennika przez } 3)$, pogrubiona+pochylona, Monospaced, kolor czcionki nr_z_dzienika +45
- Dozwolona edycja obu pól
- widoczny pasek przewijania w polu tekstowym
- do następnego wiersza można przenosić nie tylko jeden znak, lecz także całe słowo w polu tekstowym.
- Ustawienie rozmiar tabulacji na 6 w polu tekstowym.
- ustalenie wartości początkowej pozycji kursora na 3 w polu tekstowym.

Przyciski - klasa Button

Przyciski są obiektami klasy JButton. Deklaracja przycisku wraz z tekstem znajdującym się na nim jest następująca:

```
JButton przycisk=new JButton("jakiś napis");
```

Jednak samo utworzenie przycisku nic nam nie daje, dopóki nie stworzymy obiektu odpowiedzialnego za obsługę zdarzeń.

Algorytm wykorzystania przycisku JButton w programie

Krok1

należy załadować dwie klasy:

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

Krok2

należy utworzyć nowy obiekt, który należy do klasy ActionListener

Zdarzeniem w tym przypadku będzie wciśnięcie przycisku. Obiekt oczekujący na zdarzenie należy do klasy ActionListener i definiuje się go w sposób pokazany poniżej:

```
ActionListener radar=new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        /* instrukcje do wykonania
        gdy przycisk jest wciśnięty */
    }
}
```

Obiekt radar posiada metodę actionPerformed, która wywoływana jest, gdy zachodzi określone zdarzenie.

Jeden obiekt wykrywający zdarzenie może być przypisany do wielu przycisków, dlatego argument ActionEvent może w takiej sytuacji posłużyć do ich rozróżnienia.

Co więcej do jednego obiektu (przycisku) może być przypisane wiele obiektów reagujących na różne lub te same zdarzenia?

Krok3

należy dołączyć obiekt do przycisku

Po utworzeniu obiektu klasy ActionListener należy go jeszcze dołączyć do przycisku, który ma obsługiwać, używając metody addListener() :

```
przycisk.addListener(radar);
```

Krok4

należy włączyć przycisk wydając polecenie:

```
przycisk.setEnabled(true);
```

Krok5

należy dodać do panelu:

```
panel.add(przycisk);
```

kilku zdarzeń związanych z myszą tj.:

mousePressed - kliknięcie myszka na obiekcie

mouseReleased - puszczenie przycisku myszki nad obiektem

mouseEntered - najechanie kursorem na obiekt

mouseExited - kursor myszki opuszcza obszar obiektu

Uwaga:

Przyciski Button (JButton) można wykonać z użyciem biblioteki:

- import java.awt.*; → definiowanie Button
- import javax.swing.*; → definiowanie JButton

Przykład 28

- Nazwa klasy to nazwisko_ucznia_p28 dla pierwszego pliku (zamiast bat).

- Treść zadania:

Program, który po naciśnięciu przycisku powodować będzie:

- wyświetlanie napisu (w okienku DOS) gdy wciskamy przycisk, zanotuj w zeszycie jaki to napis?
- najeżdżamy na przycisk kursorem myszy, zanotuj w zeszycie jaki napis pojawia się?
- kursor myszy opuszcza obszar przycisku, zanotuj w zeszycie jaki napis pojawia się?
- Zanotuj w zeszycie temat (3 podpunkty)

Temat:

- Demonstracja użycia przycisków JButton
- Podłączenie do przycisku JButton nasłuchu czy klawisz jest wciśnięty
ActionListener radar=new ActionListener()
- Uruchomienie nasłuchu myszki
MouseAdapter mysz=new MouseAdapter()

Musisz stworzyć dwa pliki:

Plik1

Najpierw program główny, zawierający metodę main():

```
class bat
{
    public static void main(String[] args)
    {
        panel nowy_panel=new panel();
    }
}
```

Plik 2

i klasa, w której definiujemy interfejs graficzny z obsługą zdarzeń:

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

public class panel
```

```

{
//bezparametrowy konstruktor panelu
public panel()
{
    JFrame okno=new JFrame();
    okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    okno.setVisible(true);
    okno.setSize(400,200);
    JPanel p=new JPanel();
    p.setLayout(new FlowLayout());
    p.setPreferredSize(new Dimension(400,200));
    p.setBackground(Color.blue);
//-----pole tekstowe-----//
    final JTextField pole=new JTextField("pole tekstowe",30);
    p.add(pole);
/* W siedmiu liniach tworzony jest przycisk wraz z obiektem reagującym na zdarzenie
(wciśnięcie przycisku), który następnie dodajemy do panelu.*/
    JButton przycisk=new JButton("nacisnij");
    ActionListener radar=new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            System.out.println("halo");
        }
    };
    przycisk.addActionListener(radar);
    przycisk.setEnabled(true);
//-----//
/* kilku zdarzeń związanych z myszą tj.:
mousePressed - kliknięcie myszka na obiekcie
mouseReleased - puszczenie przycisku myszki nad obiektem
mouseEntered - najechanie kursorem na obiekt
mouseExited - kursor myszki opuszcza obszar obiektu */

    MouseAdapter mysz=new MouseAdapter()
    {
        public void mousePressed(MouseEvent e)
        {
            pole.setText("wcisnales przycisk");
        }
        public void mouseReleased(MouseEvent e)
        {
            pole.setText("pusciles przycisk");
        }
        public void mouseEntered(MouseEvent e)
        {
            pole.setText("wjechales na przycisk");
        }
        public void mouseExited(MouseEvent e)
        {

```

```

        pole.setText("wyjechales poza przycisk");
    }
};
//-----//
/* do przycisku przypisywany (rejestrowany) jest obiekt reagujący na zdarzenia wywołane
przez mysz. */

przycisk.addMouseListener(mysz);
/*do panelu dodawany (rejestrowany) jest przycisk*/

p.add(przycisk);
okno.setContentPane(p);
okno.pack();
}
}

```

Opis-wniosek

Jeśli jedno z tych zdarzeń zajdzie to w polu tekstowym wyświetli się odpowiedni komunikat. Najważniejsze jest jednak to, że obiekt obsługujący zdarzenia generowane przez myszkę może być przypisany do innych niż przyciski obiektów. Oprócz opisanych powyżej zdarzeń istnieje też grupa zdarzeń reagujących na sam ruch myszki.

Zdarzenia związane z oknem

Z każdym oknem związany jest zestaw zdarzeń reprezentowanych przez **interfejs WindowListener**.

Interfejs ten definiuje metody, które zostały zebrane w tabeli poniżej. Oczywiście sama implementacja interfejsu to nie wszystko, musimy jeszcze poinformować system, że to właśnie nasze okno ma odbierać wysyłane komunikaty, co robimy, wywołując metodę `addWindowListener`.

Metody interfejsu WindowListener

Typ zwracany	Nazwa metody	Opis
Void	<code>windowActivated(WindowEvent e)</code>	Metoda wywoływana po aktywacji okna
Void	<code>windowClosed(WindowEvent e)</code>	Metoda wykonywana, kiedy okno zostanie zamknięte poprzez wywołanie metody <code>dispose</code>
Void	<code>windowClosing(WindowEvent e)</code>	Metoda wywoływana, kiedy następuje próba zamknięcia okna przez użytkownika.
Void	<code>windowDeactivated(WindowEvent e)</code>	Metoda wywoływana po dezaktywacji okna.
Void	<code>windowDeiconified(WindowEvent e)</code>	Metoda wywoływana, kiedy okno zmieni stan ze zminimalizowanego na normalny.
Void	<code>windowIconified(WindowEvent e)</code>	Metoda wywoływana po minimalizacji okna
Void	<code>windowOpened(WindowEvent e)</code>	Metoda wywoływana po otwarciu okna

Przykład 29

Wykonaj:

- Uruchom przykład lecz pamiętaj aby nadać: Nazwa klasy to nazwisko_ucznia_p29 dla pierwszego pliku (zamiast Aplikacja).
- Przepisz do zeszytu temat (6 podunktów)

Temat:

- Demonstracja użycia przycisków Button
- Podłączenie do przycisku Button nasłuchu czy klawisz jest wciśnięty
ActionListener radar=new ActionListener()
- Uruchomienie nasłuchu co dzieje się z oknami systemowymi (zamknięcie itp.
addWindowListener(new WindowAdapter())
- Wykonanie okna Dialogowego
JOptionPane.showMessageDialog(null,"Technik informatyk","O jestem",1);
- Rozpoznanie, który przycisk jest wciśnięty
if(e.getActionCommand().equals("b_wyjście"))
- Definiowanie wielkości przycisku
button1.setBounds(100, 40, 160, 20);

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

public
class Aplikacja extends Frame
{

    public Aplikacja()
    {
        ActionListener al = new ActionListener() //włączenie nasłuchu myszki
        {
            public void actionPerformed(ActionEvent e)
            {
                if(e.getActionCommand().equals("b_wyjście"))
                {
                    dispose();
                } // zwolnienie pamięci
                if(e.getActionCommand().equals("b_niespodzianka"))
                {
                    JOptionPane.showMessageDialog(null,"Technik informatyk","O jestem",1);}
            }
        };

        addWindowListener(new WindowAdapter() //włączenie nasłuchu co się dzieje z
oknami
        {
            public void windowClosing(WindowEvent e)
            {
                dispose();
            }
        });
    }
}
```

```

        System.exit(0);
    }
});
setLayout(null);
Button button1 = new Button("Koniec programu");
button1.setBounds(100, 40, 160, 20);
button1.addActionListener(al);
button1.setActionCommand("b_wyjscie");

Button button2 = new Button();
button2.setLabel("Niespodzianka wcisnij");
button2.setBounds(100, 80, 160, 20);
button2.addActionListener(al);
button2.setActionCommand("b_niespodzianka");

add(button1);
add(button2);
setSize( 400, 200);
setVisible(true);
}

public static void main(String args[])

    {
        new Aplikacja();
    }

}

```

Koniec przykładu 29

Przykład 30

- Uruchom przykład lecz pamiętaj aby nadać: Nazwa klasy to nazwisko_ucznia_p30 dla pierwszego pliku (zamiast Aplikacja).

- **Dopisać zmianę coś z oknami i obliczeniami**

- Przepisz do zeszytu temat.

- Zapisz w zeszycie dane wejściowe do programu (zaproponowane przez ucznia) oraz dane wyjściowe. Zastosuj następujący zapis w zeszycie:

Dane wejściowe:

.....

Dane wyjściowe:

.....

Temat: Zamiana ułamka na dziesiętny.

```

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

```

```

public

```

```

class Aplikacja extends Frame
{

    public Aplikacja()
    {
        ActionListener al = new ActionListener() //właczenie nasluchu myszki
        {
            public void actionPerformed(ActionEvent e)
            {
                if(e.getActionCommand().equals("b_wyjście"))
                {
int wybor = JOptionPane.showConfirmDialog(null,"Zamknąć program?", "Zamykanie programu",2);
                    if(wybor == JOptionPane.OK_OPTION)
                    {
                        dispose();
                    }
                    if(wybor==JOptionPane.OK_CANCEL_OPTION)
                    {
                        return;
                    }
                }
                if(e.getActionCommand().equals("b_obliczenia"))
                {
                    String x1,x2;
                    x1=JOptionPane.showInputDialog(null,"Podaj licznik");
                    x2=JOptionPane.showInputDialog(null,"Podaj mianownik");
                    float l1=Float.parseFloat(x1);
                    float l2=Float.parseFloat(x2);
                    if (l2==0)
                    {
                        JOptionPane.showMessageDialog(null," dzielenie przez zero","ZSE",1);
                    }
                    else
                    {
                        float iloraz=(float)l1/l2;
                        JOptionPane.showMessageDialog(null,""+iloraz,"ZSE",1);
                    }
                }
            }
        };

        addWindowListener(new WindowAdapter() //właczenie nasluchu co sie dzieje z oknami
        {
            public void windowClosing(WindowEvent e)
            {
                dispose();
                System.exit(0);
            }
        }
    }
}

```

```

    });

    setLayout(null);

    Button button1 = new Button("Koniec programu");
    button1.setBounds(100, 40, 160, 20);
    button1.addActionListener(al);
    button1.setActionCommand("b_wyjście");

    Button button2 = new Button();
    button2.setLabel("Obliczenia");
    button2.setBounds(100, 80, 160, 20);
    button2.addActionListener(al);
    button2.setActionCommand("b_obliczenia");

    add(button1);
    add(button2);
    setSize(400, 200);
    setVisible(true);
}

public static void main(String args[])

    {
        new Aplikacja();
    }
}

```

Koniec przykładu 30

Zadanie na przykład 20b proste obliczenie dla każdego ucznia inne. Teoria do okienek informacyjnych i obliczeniowych.

Teoria z CheckboxGroup Choice Checkbox.

Przykład 31

- Uruchom przykład lecz pamiętaj aby nadać: Nazwa klasy to nazwisko_ucznia_p20c dla pierwszego pliku (zamiast Aplikacja).
- Przepisz do zeszytu temat (cztery podpunktów).

Temat:

- Demonstacja użycia wyjątków Instrukcja **try ... catch....**
- Demonstacja użycia **CheckboxGroup** → Przyciski opcji - pozwalają wybrać jedną z wielu opcji.
- Demonstacja użycia **Choice** → lista rozwijalna, pozwalają wybrać jedną z wielu opcji.
- Demonstacja użycia **Checkbox** → Pola wyboru - pozwalają włączać lub wyłączać różne opcje, może być wybrane wiele opcji.

Wykonaj:

- Nazwa klasy to nazwisko_ucznia_p20c dla pierwszego pliku (zamiast Aplikacja).
- Po uruchomieniu programu i przetestowaniu, zmień stawki Vat na:
 - Numer_miesiąca urudzenia+1
 - Numer_miesiąca urudzenia+3
 - Numer_miesiąca urudzenia+5
 - Numer_miesiąca urudzenia+7
 - Numer_miesiąca urudzenia+12
- Pamiętaj o zmianie wzorów na obliczenia z nowym Vatem oraz napisów na liście rozwijalnej.
- Dopisz czwartą opcję pola wyboru (Checkbox) „Ubezpieczenie” jako jeden procent wartości (są już w programie „dostawa do domu” „kurierem” „płatność przy odbiorze”).
Zmienić(dopisać) tak aby ta opcja była w różnych miejscach w zależności od nr w dzienniku.
- Będziesz musiał poprzysować w inne miejsce okna elementy tak aby zrobić miejsce na wpisany element Ubezpieczenie.

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

```
public
class Aplikacja extends Frame
{
    public TextField textFieldcenanetto;
    public TextField textFieldwynik;
    public CheckboxGroup userCheckbox1;
    public double l2;
    public Choice lista_vat;
    public Checkbox cb1;
    public Checkbox cb2;
    public Checkbox cb3;

    public Aplikacja()
    {

        addWindowListener(new WindowAdapter() //włączenie nasluchu co sie dzieje z oknami
        {
            public void windowClosing(WindowEvent e)
            {

                dispose();
                System.exit(0);

            }
        }
    }
}
```



```

        {
            l2=l2*1.23;
        }
// pobranie który checkbox jest wybrany
        if(cb1.getState() == true)
        {
            l2=l2*1.1;
        }
        if(cb2.getState() == true)
        {
            l2=l2*1.30;
        }
        if(cb3.getState() == true)
        {
            l2=l2*1.2;
        }

        textFieldwynik.setText(""+l2);

    }
    if(string_opakowanie.equals("czy bez opakowania ?"))
    {
        l2=11;
// pobranie która opcja z listy jest wybrana

        if(string_lista.equals("0 % VAT"))
        {
            l2=l2*1;
        }
        if(string_lista.equals("3 % VAT"))
        {
            l2=l2*1.03;
        }
        if(string_lista.equals("5 % VAT"))
        {
            l2=l2*1.05;
        }
        if(string_lista.equals("7 % VAT"))
        {
            l2=l2*1.07;
        }
        if(string_lista.equals("23 % VAT"))
        {
            l2=l2*1.23;
        }
// pobranie który checkbox jest wybrany
        if(cb1.getState() == true)
        {
            l2=l2*1.1;
        }

```

```

        if(cb2.getState( )==true)
        {
            l2=l2*1.30;
        }
        if(cb3.getState( )==true)
        {
            l2=l2*1.2;
        }
// wypisanie wyniku
        textFieldwynik.setText(""+l2);
    }

    }
catch (NumberFormatException n) // poczatek bloku gdy jest blad
    // konwersji na liczbe
    {
        return;
    }
}

};

setLayout(null);

textFieldcenanetto=new TextField("");
textFieldcenanetto.setBounds(180,130,50,20);
//setBounds(int x, int y, int width, int height).

Label labelcena=new Label("Podaj cene netto");
labelcena.setBounds(50,130,120,20);

textFieldwynik=new TextField("");
textFieldwynik.setBounds(180,330,80,20);

Label labelwynik=new Label("Musisz zapłacić");
labelwynik.setBounds(50,330,120,20);

userCheckbox1=new CheckboxGroup();
Checkbox checkbox_opakowanie=new Checkbox("czy w opakowaniu ?",userCheckbox1,false);
checkbox_opakowanie.setBounds(50,170,150,15);
Checkbox checkbox_bez_opakowania=new Checkbox("czy bez opakowania ?",userCheckbox1,true);
checkbox_bez_opakowania.setBounds(50,200,150,15);

cb1=new Checkbox("dostawa do domu");
cb1.setBounds(50,230,150,15);
cb2=new Checkbox("kurierem");
cb2.setBounds(50,250,150,15);
cb3=new Checkbox("platnosc przy odbiorze");

```



```

        cb3.setBounds(50,270,150,15);

        lista_vat=new Choice();
        lista_vat.setBounds(50,290,150,15);
        lista_vat.addItem("0 % VAT");
        lista_vat.addItem("3 % VAT");
        lista_vat.addItem("5 % VAT");
        lista_vat.addItem("7 % VAT");
        lista_vat.addItem("23 % VAT");

        Button button1 = new Button("Koniec programu");
        button1.setBounds(100, 40, 160, 20);
        button1.addActionListener(al);
        button1.setActionCommand("b_wyjście");

        Button button2 = new Button();
        button2.setLabel("Obliczenia");
        button2.setBounds(100, 80, 160, 20);
        button2.addActionListener(al);
        button2.setActionCommand("b_obliczenia");

        add(lista_vat);
        add(textFieldcenanetto);
        add(labelcena);
        add(textFieldwynik);
        add(labelwynik);
        add(checkbox_opakowanie);
        add(checkbox_bez_opakowania);
        add(cb1);
        add(cb2);
        add(cb3);
        add(button1);
        add(button2);
        setSize( 400, 400);
        setVisible(true);
    }

    public static void main(String args[])

        {
            new Aplikacja();
        }

}

```

Koniec przykładu 31

Przykład 32

- Uruchom przykład lecz pamiętaj aby nadać: Nazwa klasy to nazwisko_ucznia_p32 dla pierwszego pliku.

- Przepisz do zeszytu temat oraz liczby, które wpisałeś do żółtych pól oraz jaki był wynik działania programu.

Temat: Program, który po wprowadzaniu danych do dwóch żółtych pól obliczy iloraz tych liczb po naciśnięciu przycisku oblicz.

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

class tekst1
{
    public static void main(String[] args)
    {
        JFrame okno=new JFrame();
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        okno.setVisible(true);
        JPanel p=new JPanel();
        p.setLayout(new BorderLayout());
        // kontener p w którym komponenty umieszczone elementy będą wg stron świata
        JPanel p_lewo=new JPanel();
        GridLayout rozmieszczenie_siatka = new GridLayout(2,1);
        // kontener p_lewo w którym komponenty umieszczone elementy będą wg siatki
        p_lewo.setLayout(rozmieszczenie_siatka);
        JPanel p_srodek=new JPanel();
        p_srodek.setLayout(rozmieszczenie_siatka);
        JPanel p_prawo=new JPanel();
        p_prawo.setLayout(rozmieszczenie_siatka);
        p.setPreferredSize(new Dimension(340,150));
        p.setBackground(Color.blue);
        // wielkość konteneru p oraz jego kolor tła
        String s1="Wprowadz dane do żółtych pól";
        Font czcionka=new Font("Dialog", Font.ITALIC+Font.BOLD,20);
        //ustalamy parametry czcionki wyświetlanej w polu tekstowym
        final JTextField liczba1=new JTextField(6);
        liczba1.setFont(czcionka);
        liczba1.setBackground(Color.yellow);
        final JTextField liczba2=new JTextField(6);
        liczba2.setBackground(Color.yellow);
        liczba2.setFont(czcionka);
        final JTextField wynik=new JTextField(12);
        wynik.setFont(czcionka);
        //tworzymy pole tekstowe i inicjujemy je.
        JLabel opis=new JLabel(s1);
        opis.setOpaque(true); // aby można zmienić kolor Label
        opis.setBackground(new Color(130,200,138));

        opis.setFont(czcionka);
```

```

JButton wprowadz=new JButton("oblicz");
JLabel opis_liczba1=new JLabel("pierwsza liczba");
JLabel opis_liczba2=new JLabel("druga liczba");
JLabel opis_wynik=new JLabel("wynik dzielenia");
opis_wynik.setOpaque(true);
opis_wynik.setBackground(new Color(192,100,18));

ActionListener akcja= new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("wykonanie programu");
    }
};
wprowadz.addActionListener(akcja);
wprowadz.setEnabled(true);
MouseAdapter mysz=new MouseAdapter()
{
    public void mousePressed(MouseEvent e)
    {
        if (liczba1.getText()!=null && liczba2.getText()!=null
            && !liczba1.getText().equals("") && !liczba2.getText().equals("")
            && !liczba2.getText().equals("0"))
        {
            float l1=Float.parseFloat(liczba1.getText());
            float l2=Float.parseFloat(liczba2.getText());
            float iloraz=(float)l1/l2;
            wynik.setText(""+iloraz);
        }
    }
};

wprowadz.addMouseListener(mysz);
// do kontenera p_lewo dodajemy dwa komponenty
p_lewo.add(opis_liczba1);
p_lewo.add(liczba1);

p_prawo.add(opis_liczba2);
p_prawo.add(liczba2);

p_srodek.add(opis_wynik);
p_srodek.add(wynik);

p.add(BorderLayout.NORTH,opis);
p.add(BorderLayout.EAST,p_prawo);
p.add(BorderLayout.WEST,p_lewo);
p.add(BorderLayout.SOUTH,wprowadz);
p.add(BorderLayout.CENTER,p_srodek);

okno.setContentPane(p);

```

```
        okno.pack();  
    }  
}
```

Zadanie 29 (zadanie nieobowiązkowe. Przeznaczone dla uczniów, którzy chcą dostać 5 lub 6→można pominąć, będziesz mógł wrócić na koniec wykonywania tej tabelki)

Temat Program rozwiązujący równanie kwadratowe po podaniu A,B,C.

Układ okna wg numeru z dziennika.

Możliwe teksty:

- Dwa rozwiązania,
- Jedno rozwiązanie,
- Brak rozwiązań.

(Dla nr_w_dzienniku) mod 4=0		
NAZWISKO Ucznia		
A	Tekst	X0= Gdy brak to —
B		X1= Gdy brak to —
C		X2= Gdy brak to —

(Dla nr_w_dzienniku) mod 4=1		
NAZWISKO Ucznia		
X0= gdy brak to —	X1= gdy brak to —	X2= Gdy brak to —
	Tekst	
A	B	C

(Dla nr_w_dzienniku) mod 4=2		
NAZWISKO Ucznia		
X0= Gdy brak to —		A
X1= Gdy brak to —	Tekst	B
X2= Gdy brak to —		C

(Dla nr_w_dzienniku) mod 4=3		
NAZWISKO Ucznia		
A	B	C
	Tekst	
X0= Gdy brak to —	X1= gdy brak to —	X2= Gdy brak to —

- Klasa zawierająca main ma mieć nazwę Nazwisko ucznia (bez polskich liter).
- Wielkość okna 600+nr_z_dziennika, 400+nr_z_dziennika
- Tytuł okna nazwisko ucznia.
- Klasa pochodną klasy JPanel ma mieć nazwę Pan_nazwisko_ucznia.

- Ramka grubość numer_z_dziennika, kolor zielony

Pola JCheckBox

Klasa JCheckBox tworzy komponenty będące polami wyboru umożliwiającymi zaznaczanie opcji.

Konstruktor	Opis
JCheckBox()	Tworzy niezaznaczone pole wyboru, bez ikony i przypisanego tekstu
JCheckBox(Action a)	Tworzy nowe pole wyboru o właściwościach wskazanych przez argument a.
JCheckBox(Icon icon)	Tworzy niezaznaczone pole wyboru z przypisaną ikoną
JCheckBox(Icon icon, boolean selected)	Tworzy pole wyboru z ikoną, jego stan jest określony przez argument selected.
JCheckBox(String text)	Tworzy niezaznaczone pole wyboru z przypisanym tekstem.
JCheckBox(String text, boolean selected)	Tworzy pole wyboru z przypisanym tekstem, którego stan jest określony przez argument selected.
JCheckBox(String text, Icon icon)	Tworzy niezaznaczone pole wyboru z przypisaną ikoną oraz tekstem.
JCheckBox(String text, Icon icon, boolean selected)	Tworzy pole wyboru z przypisaną ikoną i tekstem, którego stan jest określony przez argument selected.

KOMPONENT CHECKBOX

Checkbox - inaczej pola wyboru to komponenty służące do zaznaczania i odznaczania określonych opcji.

Checkbox jest klasą zawierającą pięć wersji konstruktorów:

- * Checkbox() → Domyślny konstruktor, tworzy pusty komponent.
- * Checkbox(String str) → Ta wersja konstruktora tworzy checkbox z napisem przekazanym w parametrze str.
- * Checkbox(String str, boolean zaznaczony) → Dodatkowy argument ustala czy komponent ma być zaznaczony.
- * Checkbox(String str, boolean zaznaczony, CheckboxGroup cg)
Ostatni parametr decyduje do jakiej grupy komponent będzie przynależeć.
- * Checkbox(String str, CheckboxGroup cg, boolean zaznaczony)
To samo co wyżej tylko w innej kolejności.

Tekst Checkboxa możemy ponadto pobrać metodą getLabel(),
i zmienić metodą setLabel(String str).

Do zmiany zaznaczenia służy metoda setState(boolean zaznaczony), przy czym wartość true ustawia zaznaczenie, a false je anuluje.

Żeby sprawdzić czy dany Checkbox jest zaznaczony możemy posłużyć się metodą getState(), która zwraca wartość typu boolean.

Gdy Checkboxy są zgrupowane zaznaczony może być tylko jeden Checkbox, a zaznaczenie innego spowoduje odznaczenie poprzedniego.

Oto kod, obrazujący w jaki sposób dodaje się Checkboxa który nie jest zgrupowany:

```
Checkbox cb = new Checkbox("checkbox1");  
add(cb);
```

A oto przykład zgrupowanych Checkboxów:

```
CheckboxGroup cg = new CheckboxGroup( );  
Checkbox cb1 = new Checkbox("checkbox1", cg, true);  
Checkbox cb2 = new Checkbox("checkbox2", cg, false);  
Checkbox cb3 = new Checkbox("checkbox3", cg, false);  
Checkbox cb4 = new Checkbox("checkbox4", cg, false);  
add(cb1);  
add(cb2);  
add(cb3);  
add(cb4);
```

Aby obsłużyć zdarzenia należy zaimplementować interfejs ItemListener.

Interfejs ten definiuje tylko jedną metodę: itemStateChanged(ItemEvent e).

Jest ona wywoływana gdy zaznaczymy lub odznaczymy Checkboxa.

Musimy także wywołać metodę addItemListener, jako argument podając klasę obsługującą zdarzenie.

Klasa CheckboxGroup ma jedną dość istotną metodę, mianowicie getSelectedCheckbox(), która zwraca klasę Checkbox z aktualnie zaznaczonym komponentem.

Przykład 32

Wykonaj:

- Nazwa klasy to nazwisko_ucznia_p21a dla pierwszego pliku (zamiast Aplikacja).

1)Przepisz temat

2)Uruchom przykład

3)Zmień przykład tak aby można było zaznaczyć 8 opcji. Opcje to kraje z którymi graniczy Polska oraz morze Bałtyk (jako osobna opcja).

Może dopisać akcje jakie są zaznaczone opcje i wypisać „Które kraje chcesz zwiedzić” po zaznaczeniu zostanie wyświetlona lista tych krajów. Napisac jak to zrobić tak aby uczeń musiał to zrobić wzoruj się na 20c, można zmienić kolejnościa 21a z 21b. Może przenieść przed 20c.

Temat: Aplikacja zawierającą sześć (osiem) pól wyboru.

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

```

public
class Aplikacja extends JFrame
{
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(3 , 2));
        JCheckBox cb1 = new JCheckBox("Opcja 1");
        JCheckBox cb2 = new JCheckBox("Opcja 2");
        JCheckBox cb3 = new JCheckBox("Opcja 3");
        JCheckBox cb4 = new JCheckBox("Opcja 4");
        JCheckBox cb5 = new JCheckBox("Opcja 5");
        JCheckBox cb6 = new JCheckBox("Opcja 6");
        add(cb1);
        add(cb2);
        add(cb );
        add(cb4);
        add(cb5);
        add(cb6);
        setSize( 200, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Nie podajemy bowiem żadnych współrzędnych, w których mają znaleźć się poszczególne elementy okna. Zamiast tego zastosowaliśmy jeden z rozkładów automatycznych, tzw. rozkład tabelaryczny lub siatkowy (ang. grid layout). Za jego ustawienie odpowiada linia: `setLayout(new GridLayout(3 , 2));`

Parametrem metody `setLayout` jest nowy obiekt klasy `GridLayout`. Dzięki temu powierzchnia okna aplikacji zostanie podzielona na sześć części, tabelę o dwóch wierszach i trzech kolumnach. Komponenty, które będą dodawane do okna aplikacji, będą od tej chwili automatycznie umieszczane w kolejnych komórkach takiej tabeli, dzięki czemu uzyskamy ich równomierny rozkład.

Listy rozwijane

Tworzenie list rozwijalnych umożliwia klasa o nazwie `JComboBox`.

Konstruktor	Opis
JComboBox()	Tworzy pustą listę
JComboBox(ComboBoxModel aModel)	Tworzy nową listę z modelem danych wskazanym przez argument aModel.
JComboBox(Object[] items)	Tworzy listę zawierającą dane znajdujące się w tablicy items.
JComboBox(Vector<?> items)	Tworzy listę zawierającą dane znajdujące się w wektorze items

Wybrane metody klasy JComboBox.

Deklaracja metody	Opis
void addItem(Object anObject)	Dodaje nową pozycję do listy.
Object getItemAt(int index)	Pobiera element listy znajdujący się pod wskazanym indeksem.
int getItemCount()	Pobiera liczbę elementów listy.
int getSelectedIndex()	Pobiera indeks zaznaczonego elementu.
Object getSelectedItem()	Pobiera zaznaczony element.
void insertItemAt(Object anObject, int index)	Wstawia nowy element we wskazanej pozycji listy.
boolean isEditable()	Zwraca true, jeśli istnieje możliwość edycji listy.
void removeAllItems()	Usuwa wszystkie elementy listy.
void removeItem(Object anObject)	Usuwa wskazany element listy.
void removeItemAt(int anIndex)	Usuwa element listy znajdujący się pod wskazanym indeksem.
void setEditable(boolean aflag)	Ustala, czy lista ma mieć możliwość edycji.
void setSelectedIndex(int anIndex)	Zaznacza element listy znajdujący się pod wskazanym indeksem.
void setSelectedItem(Object anObject)	Zaznacza wskazany element listy.

Przykład 33

Potem zadanie na jakąś akcję na wybór opcji może ten przykład przerobić aby była akcja przy wyborze z listy sprawdzić jak połączyć z 20c → przestawić kolejność może?

Wykonaj:

- Nazwa klasy to nazwisko_ucznia_p21b dla pierwszego pliku (zamiast Aplikacja).

1)Przepisz temat

2)Zapisz działanie programu

3)Uruchom przykład

4)Zmień przykład tak aby lista rozwijalna wyświetlała się nad etykietą.

5)Zmień przykład tak aby lista rozwijalna miała opcje:

-Nazwisko ucznia

-Imię ucznia

-ulubiony aktor

-ulubiony sport

-ulubiona potrawa
-numer w dzienniku

Temat: Aplikacja zawierająca jedną listę rozwijalną oraz jedną etykietę.

Działanie programu: Po wybraniu nowej pozycji z listy przypisany jej tekst pojawi się na etykiecie.

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

public
class Aplikacja extends JFrame
{
    JComboBox cmb;
    JLabel label;
    public Aplikacja()
    {
        ActionListener al = new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                int itemIndex = cmb.getSelectedIndex();
                if(itemIndex < 1) return;
                String itemText = cmb.getSelectedItem().toString();
                label.setText(itemText);
            }
        };
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        label = new JLabel("Wybierz pozycję z listy.");
        label.setBounds(80, 10, 170, 20);
        cmb = new JComboBox();
        cmb.setBounds(80, 40, 170, 20);
        cmb.addItem("Wybierz książkę...");
        cmb.addItem("Java. Ćwiczenia praktyczne");
        cmb.addItem("Praktyczny kurs Java");
        cmb.addItem("Java. Leksykon kieszonkowy");
        cmb.addActionListener(al);
        add(cmb);
        add(label);
        setSize(400, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
```

```
        new Aplikacja();
    }
});
}
}
```

Elementy listy są dodawane za pomocą metody `addItem` klasy `JComboBox`. Zdarzenia nadchodzące z listy obsługuje obiekt klasy anonimowej implementującej interfejs `ActionListener` o nazwie `al`. Jest w nim oczywiście zdefiniowana tylko jedna metoda **`actionPerformed`**. Za pomocą wywołania `cmb.getSelectedIndex()` pobiera ona najpierw indeks zaznaczonego elementu listy. Jeśli okaże się, że indeks ten jest mniejszy od 1 (co by oznaczało, że albo nie został zaznaczony żaden z elementów, albo też zaznaczony jest element o indeksie 0), metoda kończy działanie, wywołując instrukcję `return`. Jeśli indeks jest większy od 0, pobierany jest tekst przypisany zaznaczonemu elementowi listy: `String itemText = cmb.getSelectedItem().toString();` który następnie jest wykorzystywany jako argument metody `setText` obiektu etykiety: `label.setText(itemText);`

Instrukcja eclipse

Eclipse to rozbudowane środowisko programistyczne (framework) stworzone przez firmę IBM napisane w języku Java i przekazane następnie społeczności Open Source (otwarty kod). Eclipse ma możliwość obsługi wtyczek rozszerzających jego możliwości o obsługę wielu języków programowania. Aby móc korzystać z wszystkich możliwości Eclipse musisz dograć wtyczki rozszerzające nazywane pluginami.

***Framework** to inaczej platforma programistyczna, szkieletem do budowy aplikacji. Definiuje on strukturę aplikacji oraz ogólny mechanizm jej działania, a także dostarcza zestaw komponentów i bibliotek ogólnego przeznaczenia do wykonywania określonych zadań. Programista tworzy aplikację, rozbudowując i dostosowując poszczególne komponenty do wymagań realizowanego projektu, tworząc w ten sposób gotową aplikację.*

Możliwości eclipse:

- weryfikacja błędów,
- auto uzupełnianie kodu
- podświetlanie składni.
- po dograniu odpowiednich pluginów uzyskujesz wspomaganie pisania aplikacji w językach:
 - Java,
 - C, dzięki wtyczce CDT
 - C++, dzięki wtyczce CDT
 - PHP,
 - JavaScript,
 - Actionscript 2,
 - AmigaDT,
 - tworzenie z użyciem GUI (graficzny interfejs użytkownika → przeciąganie obiektów na formę i definiowanie właściwości obiektu).

Uwagi:

1. Program został napisany w Javie dzięki czemu jest bardzo elastyczny i można go uruchomić na różnych platformach systemowych (Windows, Linux).
2. W skład platformy, prócz IDE (edytor), wchodzi między innymi takie narzędzia, jak:
 - Web Tools Platform Project do budowania usług i aplikacji sieciowych,
 - C/C++ Development Tooling do rozwijania aplikacji w C/C++,
 - kompletne narzędzie do raportowania Business Intelligence and Reporting Tools,
 - generator kodu Eclipse Modeling Framework
 - Graphical Editing Framework do tworzenia graficznych interfejsów użytkownika.
3. Program w wersji Classic, przeznaczonej dla programistów piszących programy w języku Java. Producent wydał również pakiety dla pozostałych języków, które można pobrać ze strony pobierania.
4. Zbiór wtyczek (pluginów) do aplikacji Eclipse dostępny jest na specjalnie przygotowanej stronie.
5. Aby program Eclipse działał poprawnie wymaga jest zainstalowanych w systemie bibliotek Java 2 Runtime Enviroment (popularnie instalacja Javy).

Przygotowanie Eclipse do użycia:

1)Ściągnij Eclipse z adresu: <http://www.eclipse.org/downloads/> wersja: Eclipse IDE for Java EE Developers lub Eclipse Classic.

Eclipse jest spakowany, wystarczy go rozpakować do dowolnego folderu i jest gotowy do użycia, nie wymaga instalacji.

2)Instalowanie w systemie bibliotek Java 2 Runtime Enviroment np. z:

<http://www.dobreprogramy.pl/Java,Program,Windows,13134.html>

Środowisko-interpreter nazywany również wirtualną maszyną Java (Java Virtual Machine), umożliwia uruchamianie na komputerze i przeglądarce internetowej tzw. apletów (programów) napisanych w języku Java. W praktyce jest niezbędnym elementem do uruchamiania programów napisanych w języku Java (np. OpenOffice / StarOffice) oraz np. korzystania z internetowych chatów.

3)Instalacji wtyczki do użycia GUI→programowanie okienkowe(przeciągnij i puść)

a)Instalacji JIglloo

uruchom Eclipse→wybierz folder do którego zapisywane będą efekty Twojej pracy (okno Workspace Launcher)→

wybierz opcję z menu górnego Help →Install new software ... → w okienku Work with wpisz

<http://cloudgarden1.com/update-site> --> teraz przycisk Add-->w okienku Name wpisz nazwę tak jak będzie się nazywał plugin na Twoim dysku „plugin_okienkowy” →w okienku Name zaznacz ptaszka.



→teraz przycisk Next→nastąpi instalacja→ teraz przycisk Next→zaakceptuj warunki licencji→przycisk Finish→nastąpi instalacja (naciśnij ok. gdy nastąpią pytania o certyfikaty). --> dokonaj restartu Eclipse.

b)Instalacja WindowBuilder Pro


<https://developers.google.com/java-dev-tools/wbpro/?hl=pl>

Projektowanie w trybie GUI

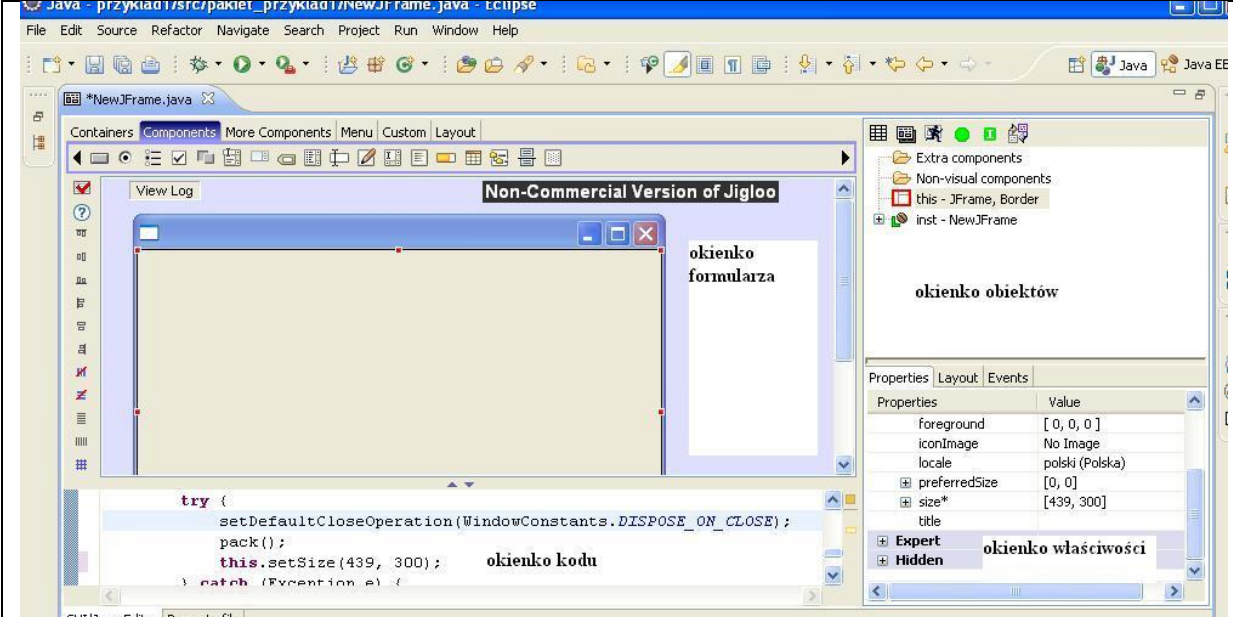
Przykład 1

Utworzenie nowego projektu GUI:

File → New → Java Project → wpisz nazwę projektu np. przyklad1 → Next (gdy nie widzisz okna Project Explorer to → Windows [z górnego menu] → Show View → Project Explorer)

	<p>Rozwiń widok tak jak obok i prawym kliknij na src → wybierz New → Other → rozwiń GUI Forms → rozwiń Swing → wybierz element JFrame → New → nazwa pakietu np. nazwisko ucznia → Finish</p>
---	--

Aby otrzymać środowisko jak poniżej zamknij okienko Eksploratora pakietów (to po lewej) i dokonaj maksymalizacji okienka formularza a otrzymasz środowisko do pracy:

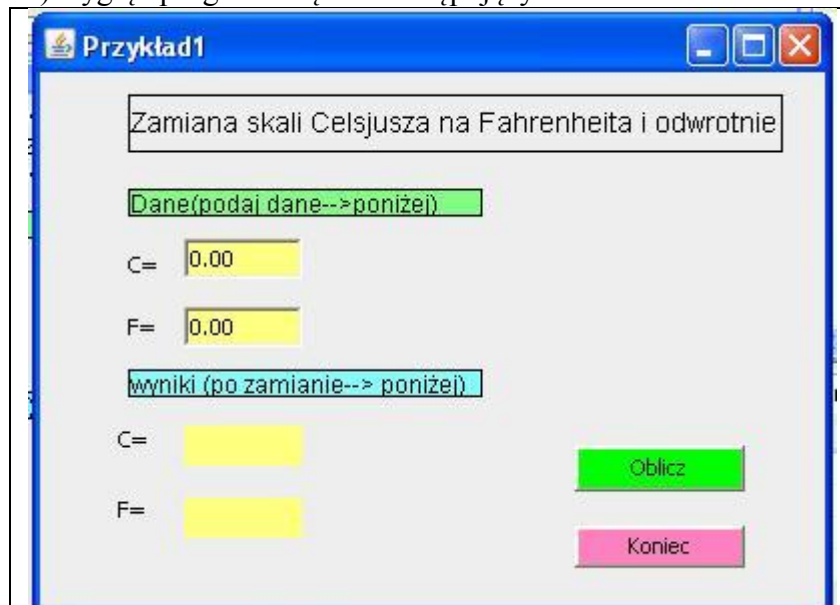


```
try {
    setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    pack();
    this.setSize(439, 300);
} catch (Exception e) {
}
```

Treść zadania:

a) Program do zamiany skali Celsjusza na Fahrenheita i odwrotnie. Program będzie miał rozszerzenie JAR.

b) Wygląd programu będzie następujący:



c) Po wykonaniu programu z jego użyciem znajdź temperaturę gdy stopnie **Celsjusza=Fahrenheita**

Rozwiązanie zadania

1) Wybierz rozkład w okienku poprzez wybór zakładki Layout i następnie



wybierz rodzaj Layout poprzez:

Kliknij w menu ikonę **AbsoluteLayout**, następnie kliknij na pole obszaru formularza.

2) Wstaw przycisk **Koniec** aby móc zakończyć program. Wykonaj zapisy poniżej:

a) W menu wybierz Components wybierz element JButton i przenieś go na formularz.

b) Zmień parametry przycisku Koniec. Czyli zmień: napis, wymiar, kolor, akcję JButton. Patrz jak ma wyglądać przycisk Koniec na widoku gotowego programu (czyli parę wierszy wyżej).

-zmiana wielkości → poprzez rozciągnięcie element JButton,

-zmian napisu → w oknie obiektów wybierz element JButton1 → w okienku właściwości wybierz zakładkę Properties i w opcji text wpisz Koniec

-zmiana koloru przycisku → w okienku właściwości wybierz zakładkę Properties i w opcji background kliknij na trzy kropki ... teraz wybierz kolor przycisku (opcja opaque na True).

-akcja do przycisku Koniec, zamknięcie programu → w okienku właściwości wybierz zakładkę Events i w opcji ActionListener → actionPerformed → handler metod Eclipse zmień kod programu wg zapisów jak poniżej:

znajdź linijki programu

```
private void jButton1ActionPerformed(ActionEvent evt) {  
    System.out.println("jButton1.actionPerformed, event="+evt);  
    //TODO add your code for jButton1.actionPerformed  
}
```

skasuj linię → `System.out.println("jButton1.actionPerformed, event="+evt);`

wpisz linię → `this.dispose();`

uwaga: `this` → jest domyślna nazwa formularza.

3) Ustaw tytuł okna na napis Przykład1 → wybierz obiekt `this-JFrame` → wybierz zakładkę Properties → `title` i tutaj wpisz tekst Przykład1.

4) Wstaw `JLabel` z napisem, kolorze (sprawdzić jak to zrobić) oraz z napisem „Zamiana skali Kelwina na Fahrenhaita i odwrotnie”

5) Wstaw pozostałe `JLabel`e (kolory tła, wielkości liter, teksty) (patrz na widok gotowego programu).

6) Wstaw wszystkie pola tekstowe `TextField` (kolory tła, wielkości liter, teksty)

7) Wykonaj obliczenia

a) akcja do przycisku Obliczenia → w okienku obiektów wybierz przycisk Obliczenia → w okienku właściwości wybierz zakładkę Events i w opcji `ActionListener` → `actionPerformed` → handler metod Eclipse wstawi kod programu uzupełnij go aby otrzymać:

```
private void jButton2ActionPerformed(ActionEvent evt) {  
    String s_c,s_f;  
    double liczba_c,liczba_f;  
    double liczba_c_obliczone,liczba_f_obliczone;  
    s_c=jTextField1.getText();  
    s_f=jTextField2.getText();  
    liczba_c=Float.parseFloat(s_c);  
    liczba_f=Float.parseFloat(s_f);  
    liczba_c_obliczone=(liczba_f-32)*5.0/9.0;  
    liczba_f_obliczone=liczba_c*(9.0/5.0)+32;  
    jLabel8.setText(String.format("%.2f",liczba_c_obliczone));  
    jLabel9.setText(String.format("%.2f",liczba_f_obliczone));  
}
```

8) Wykonanie pliku uruchomieniowego *.jar.

JAR (ang. Java ARchive) – archiwum ZIP używane do strukturalizacji i kompresji plików klas języka Java oraz powiązanych z nimi meta danych (dane o danych). Archiwum JAR składa się z pliku manifestu umieszczonego w ścieżce META-INF/MANIFEST.MF, który informuje o sposobie użycia i przeznaczeniu archiwum. Archiwum JAR, o ile posiada wyszczególnioną klasę główną, **może stanowić osobną aplikację**.

Wykonanie:

File → Eksport → Java → JAR file → Next → w okienku Select to resources to export: zaznacz okienko (pojawi się prostokąt) przed nazwą zadania w tym przypadku *przyklad1* → wybierz gdzie ma być zapisany plik **jar** w okienku JAR file: możesz użyć przycisku Browse → Next → Next → w okienku Main class: wybierz klasę główną, użyć przycisku Browse → w okienku pojawi się *kowalski.NewJFrame* → Finish → OK.

Uruchom aplikację *.jar

Tworzenie aplikacji

Przykład

Zobaczmy teraz, jak napisać aplikację, która będzie wykonywała to samo zadanie, czyli wyświetli napis na ekranie.

Wymagać to będzie napisania klasy np. o nazwie `PierwszaAplikacja`, która będzie dziedziczyć z klasy `JFrame`. Jest to klasa zawarta w pakiecie `javax.swing`. Alternatywnie można użyć również klasy `Frame` z pakietu `java.awt`, jednak jest ona uznawana za przestarzałą.

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

public
class PierwszaAplikacja extends JFrame
{
    public PierwszaAplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize( 20, 200);
        setVisible(true);
    }
    public void paint(Graphics gDC)
    {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString ("Pierwsza aplikacja", 100, 100);
    }
    public static void main(String args[])
    {
        new PierwszaAplikacja();
    }
}
```

Za utworzenie okna odpowiada klasa `PierwszaAplikacja`, która dziedziczy z klasy **`JFrame`**. W konstruktorze za pomocą metody **`setSize`** ustalamy rozmiary okna, natomiast za pomocą metody **`setVisible`** powodujemy, że zostanie ono wyświetlone na ekranie. Za wyświetlenie na ekranie napisu odpowiada metoda **`drawString`** klasy `Graphics`, odbywa się to dokładnie w taki sam sposób, jak w przypadku omawianych w poprzednich lekcjach apletów. Należy również zwrócić uwagę na instrukcję:

`setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`

która powoduje, że domyślnym działaniem wykonywanym podczas zamykania okna (np. gdy użytkownik kliknie przycisk zamykający lub wybierze taką akcję z menu systemowego) będzie zakończenie działania całego programu (mówi o tym stała `EXIT_ON_CLOSE`). Jeśli ta instrukcja zostanie pominięta, nie będzie można w standardowy sposób zakończyć działania aplikacji.

Obiekt klasy `PierwszaAplikacja` jest tworzony w metodzie **`paint`**, od której rozpoczyna się wykonywanie kodu.

Wraz z platformą Java2 SE5 pojawił się jednak nowy model obsługi zdarzeń dla biblioteki Swing, w którym operacje związane z komponentami (a okno aplikacji jest komponentem) nie powinny być obsługiwane bezpośrednio, ale trafiać do kolejki zdarzeń. Dotyczy to również samego uruchamiania aplikacji. Należy użyć metody **invokeLater** klasy **SwingUtilities**, która umieści nasze wywołanie w kolejce zdarzeń. Argumentem tej metody musi być obiekt implementujący interfejs Runnable, a operacja, którą chcemy wykonać, powinna się znaleźć w metodzie run tego interfejsu. Zgodnie z tym standardem metoda paint powinna mieć postać:

```
public static void main(String args[])
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new PierwszaAplikacja();
        }
    });
}
```

Ten też sposób będzie stosowany na listingach w dalszej części .

Przykład

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

public
class Aplikacja extends JFrame implements WindowListener
{
    public Aplikacja()
    {
        addWindowListener(this);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
    public void windowClosing(WindowEvent e)
    {
        dispose();
    }
}
```

```

public void windowClosed(WindowEvent e){ }
public void windowOpened(WindowEvent e){ }
public void windowIconified(WindowEvent e){ }
public void windowDeiconified(WindowEvent e){ }
public void windowActivated(WindowEvent e){ }
public void windowDeactivated(WindowEvent e){ }
}

```

Opis

Klasa Aplikacja dziedziczy z klasy JFrame i implementuje interfejs WindowListener. W konstruktorze poprzez wywołanie metody addWindowListener (z parametrem this wskazującym na obiekt aplikacji), powodujemy, że informacje o zdarzeniach będą przekazywane właśnie obiektowi aplikacji, czyli że będą wywoływane metody windowClosing, windowClosed, windowOpened, windowIconified, windowDeiconified, windowActivated, windowDeactivated z klasy Aplikacja. Ponieważ interesuje nas jedynie obsługa zdarzenia polegającego na zamknięciu okna, oprogramowujemy jedynie metodę windowClosing. Jest ona wywoływana, kiedy użytkownik próbuje zamknąć okno poprzez wybranie odpowiedniej pozycji z menu systemowego bądź też poprzez kliknięcie odpowiedniej ikony paska tytułowego okna. W takiej sytuacji wywołujemy metodę **dispose**, która powoduje zwolnienie zasobów związanych z oknem, zamknięcie okna i, jeżeli jest to ostatnie okno aplikacji, zakończenie pracy aplikacji.

Obsługa zdarzeń przez klasy anonimowe

Poprzedni przykład przedstawiał aplikację reagującą na zdarzenia związane z jej oknem. Konkretnie była to aplikacja, która kończyła swoje działanie po wybraniu przez użytkownika odpowiedniej pozycji z menu systemowego lub też kliknięciu właściwej ikony paska tytułowego. Możliwe to było dzięki implementacji interfejsu WindowListener bezpośrednio przez klasę okna. W takim jednak przypadku konieczna była deklaracja wszystkich metod klasy WindowListener, nawet tych, które nie były wykorzystywane. Zamiast tego wygodniej jest więc skorzystać z klasy adaptera, czyli specjalnej klasy zawierającej puste implementacje metod danego interfejsu. W przypadku interfejsu WindowListener jest to klasa WindowAdapter. Jeśli więc z WindowAdapter wyprowadzimy naszą własną klasę i przesłoniemy w niej wybraną metodę, nie będzie konieczności definiowania pozostałych. Taka sytuacja została zobrazowana na przykładzie poniżej.

```

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

public
class Aplikacja extends JFrame
{
    class MyWindowAdapter extends WindowAdapter
    {
        public void windowClosing(WindowEvent e)
        {
            dispose();
        }
    }
}

```

```

public Aplikacja()
{
    addWindowListener(new MyWindowAdapter());
    setSize( 20, 200);
    setVisible(true);
}
public static void main(String args[])
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new Aplikacja();
        }
    });
}
}

```

Opis

Tym razem w klasie Aplikacja została zdefiniowana klasa wewnętrzna MyWindowAdapter, pochodna od WindowAdapter, a w niej metoda windowClosing. W metodzie tej wywoływana jest natomiast metoda dispose klasy Aplikacja. Jest to możliwe, jako że klasa wewnętrzna ma dostęp do metod klasy zewnętrznej. W konstruktorze klasy Aplikacja została wywołana metoda addWindowListener i został jej przekazany w postaci argumentu obiekt klasy MyWindowAdapter (addWindowListener(new MyWindowAdapter());).

To nic innego jak informacja, że zdarzeniami związanymi z oknem będzie się zajmował właśnie ten obiekt. Tak więc całą obsługą zdarzenia zajmować się będzie teraz klasa MyWindowAdapter.

Zauważmy jednak, że ta klasa mogłaby być z powodzeniem klasą anonimową, jej nazwy w przedstawionej sytuacji tak naprawdę do niczego nie potrzebujemy.

Przykład

```

import javax.swing.*;
import java.awt.event.*;
public
class Aplikacja extends JFrame
{
    public Aplikacja()
    {
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                dispose();
            }
        });
        setSize( 20, 200);
        setVisible(true);
    }
}

```

```

    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Spójrzmy: w konstruktorze jest wywoływana metoda `addWindowListener` oznajmiająca, że zdarzenia związane z obsługą okna będą przekazywane obiektowi będącemu argumentem tej metody. Tym argumentem jest z kolei obiekt klasy anonimowej pochodnej od `WindowAdapter`. Ponieważ klasa anonimowa jest z natury rzeczy klasą wewnętrzną, ma ona dostęp do składowych klasy zewnętrznej — `Aplikacja` — i może wywołać metodę `dispose` zwalniającą zasoby i zamykającą okno aplikacji.

Podobnie możemy postąpić przy obsłudze zdarzeń związanych z myszą. Jeśli potrzebujemy implementacji interfejsu `MouseListener`, należy skorzystać z klasy `MouseAdapter`, jeśli natomiast niezbędny jest interfejs `MouseMotionListener`, należy skorzystać z klasy `MouseMotionAdapter`. Oba te adaptory zdefiniowane są w pakiecie `java.awt`. Pakiet `javax.swing` udostępnia natomiast dodatkowy adapter zbiorczy implementujący wszystkie interfejsy związane z myszą. Jest to `MouseInputAdapter`.

Powróćmy więc do kodu apletu, który pokazywał aktualne współrzędne kursora myszy, i przeróbmy go w taki sposób, aby do obsługi zdarzeń był wykorzystywany obiekt anonimowej klasy dziedziczącej z `MouseInputAdapter`. Kod takiego apletu jest widoczny na listingu poniżej (usunięta została jedynie błądka metody `mouseDragged`).

Przykład

```

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

public
class Aplet extends JApplet
{
    String tekst = "";
    public void init()
    {
        addMouseMotionListener(new MouseInputAdapter()
        {
            public void mouseMoved(MouseEvent evt)
            {
                tekst = "zdarzenie mouseMoved, ";
                tekst += "współrzędne: x = " + evt.getX() + ", ";
                tekst += "y = " + evt.getY();
            }
        });
    }
}

```

```

        repaint();
    }
}
);
}
public void paint (Graphics gDC)
{
    gDC.clearRect(0, 0, getSize().width, getSize().height);
    gDC.drawString(tekst, 20, 20);
}
}

```

Menu

Rzadko która aplikacja okienkowa może obyć się bez menu. W Javie w celu dodania menu musimy skorzystać z kilku klas: JMenuBar, JMenu i JMenuItem. Pierwsza z nich opisuje pasek menu, druga menu znajdujące się na tym pasku, a trzecia poszczególne elementy menu. Pasek menu dodajemy do okna aplikacji za pomocą metody setJMenuBar, natomiast menu do paska dodajemy za pomocą metody add.

```

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

public
class JAplikacja extends JFrame
{
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenuBar mb = new JMenuBar();
        JMenu menu1 = new JMenu("Menu 1");
        JMenu menu2 = new JMenu("Menu 2");
        JMenu menu = new JMenu("Menu ");
        mb.add(menu1);
        mb.add(menu2);
        mb.add(menu );
        setJMenuBar(mb);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        }
    }
}

```

```

        });
    }
}

```

W konstruktorze tworzymy nowy obiekt klasy JMenuBar i przypisujemy go zmiennej mb. Następnie tworzymy trzy obiekty klasy JMenu i dodajemy je do paska menu (czyli obiektu mb) za pomocą metody add. W konstruktorze klasy JMenu przekazujemy nazwy menu, czyli tekst, który będzie przez nie wyświetlany. Pasek menu dodajemy do okna przez wywołanie metody setJMenuBar.

Do tak stworzonego menu należy dodać poszczególne pozycje. Służy do tego klasa JMenuItem. Obiekt tej klasy dodajemy, stosując metodę add. Jeśli zatem każde menu utworzone w aplikacji z poprzedniego przykładu miałyby mieć po dwie pozycje, konstruktor należałoby zmodyfikować w sposób widoczny na listingu następnego przykładu.

```

public Aplikacja()
{
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JMenuBar mb = new JMenuBar();
    JMenu menu1 = new JMenu("Menu 1");
    JMenu menu2 = new JMenu("Menu 2");
    JMenu menu = new JMenu("Menu ");
    JMenuItem menuItem11 = new JMenuItem("Pozycja 1");
    JMenuItem menuItem12 = new JMenuItem("Pozycja 2");
    JMenuItem menuItem21 = new JMenuItem("Pozycja 1");
    JMenuItem menuItem22 = new JMenuItem("Pozycja 2");
    JMenuItem menuItem1 = new JMenuItem("Pozycja 1");
    JMenuItem menuItem2 = new JMenuItem("Pozycja 2");
    menu1.add(menuItem11);
    menu1.add(menuItem12);
    menu2.add(menuItem21);
    menu2.add(menuItem22);
    menu.add(menuItem1);
    menu.add(menuItem2);
    mb.add(menu1);
    mb.add(menu2);
    mb.add(menu);
    setJMenuBar(mb);
    setSize(20, 200);
    setVisible(true);
}

```

Tworzymy sześć różnych obiektów klasy JMenuItem odpowiadających poszczególnym pozycjom menu. Obiekty te dołączamy do kolejnych menu, wywołując metody add klasy JMenu. Czyli, przykładowo, instrukcja menu1.add(menuItem11); powoduje dodanie pierwszej pozycji do pierwszego menu. Po wykonaniu wszystkich instrukcji powyższego kodu każde menu będzie miało po dwie pozycje o nazwach Pozycja 1 i Pozycja 2.

Wiemy już, jak tworzyć menu, warto więc teraz zadać pytanie, w jaki sposób spowodować, aby program reagował na wybranie jednej z pozycji. Łatwo się zapewne domyślić, że trzeba będzie skorzystać z jakiegoś interfejsu typu `MenuListener` i przekazać zdarzenia do jakiegoś obiektu, być może obiektu aplikacji, być może obiektu dodatkowej klasy.

Faktycznie tak należy postąpić. Co prawda, nie istnieje interfejs o nazwie `MenuListener`, skorzystać należy zatem z interfejsu `ActionListener`. Jest w nim zdefiniowana tylko jedna metoda o nazwie `actionPerformed`.

Do reagowania na wybranie danej pozycji menu można więc zastosować technikę przedstawioną na listingu

Przykład

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

public
class Aplikacja extends JFrame
{
    private JMenuItem miZamknij;
    private ActionListener al = new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource() == miZamknij)
                dispose();
        }
    };
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenuBar mb = new JMenuBar();
        JMenu menu = new JMenu("Plik");
        miZamknij = new JMenuItem("Zamknij");
        menu.add(miZamknij);
        mb.add(menu);
        setJMenuBar(mb);
        miZamknij.addActionListener(al);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}
```

```

        }
    });
}
}

```

Opis

Aplikacja ma w tej chwili jedynie menu Plik z jedną pozycją o nazwie Zamknij. Sposób utworzenia menu jest analogiczny jak w poprzednich przykładach, z tą różnicą, że zmienna określająca pozycję menu Zamknij jest prywatnym polem klasy Aplikacja. Chodzi o to, abyśmy mieli do tej pozycji dostęp nie tylko w obrębie konstruktora, ale w całej klasie. Po utworzeniu paska menu (obiekt mb), samego menu (obiekt penu) oraz pozycji (obiekt miZamknij) dodajemy do tej pozycji obsługę zdarzeń, wywołując instrukcję:

```
miZamknij.addActionListener(al);
```

Tym samym zdarzenia związane z obsługą tego menu będą przekazywane do metody actionPerformed obiektu al. Obiekt ten jest obiektem klasy anonimowej implementującej interfejs ActionListener i został zdefiniowany na początku klasy Aplikacja. W metodzie actionPerformed sprawdzamy, czy obiektem, który wywołał zdarzenie, jest piZamknij, czyli pozycja menu o nazwie Zamknij. Jeśli tak, zamykamy okno, a tym samym całą aplikację. Aby uzyskać referencję do obiektu, który wywołał zdarzenie, wywołujemy metodę getSource obiektu klasy(ActionEvent) otrzymanego jako argument metody actionPerformed.

Kaskadowe menu

Pozycje menu można łączyć kaskadowo, uzyskując rozbudowane struktury podmenu. Utworzenie tego typu konstrukcji jest możliwe poprzez dodanie do menu innego menu, czyli przekazanie metodzie add klasy JMenu (w postaci argumentu) innego obiektu tej klasy.

```

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

public
class Aplikacja extends JFrame
{
    private ActionListener al = new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            String text = ((JMenuItem)e.getSource()).getText();
            System.out.println(text);
        }
    };
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenuBar mb = new JMenuBar();
        JMenu menu = new JMenu("Menu 1");
        JMenu submenu1 = new JMenu("Pozycja 4");
        JMenu submenu2 = new JMenu("Pozycja 8");
    }
}

```

```

        menu.add(new JMenuItem("Pozycja 1"));
        menu.getItem(0).addActionListener(al);
        menu.add(new JMenuItem("Pozycja 2"));
        menu.getItem(1).addActionListener(al);
        menu.add(new JMenuItem("Pozycja "));
        menu.getItem(2).addActionListener(al);
        submenu1.add(new JMenuItem("Pozycja 5"));
        submenu1.getItem(0).addActionListener(al);
        submenu1.add(new JMenuItem("Pozycja 6"));
        submenu1.getItem(1).addActionListener(al);
        submenu1.add(new JMenuItem("Pozycja 7"));
        submenu1.getItem(2).addActionListener(al);
        submenu2.add(new JMenuItem("Pozycja 9"));
        submenu2.getItem(0).addActionListener(al);
        submenu2.add(new JMenuItem("Pozycja 10"));
        submenu2.getItem(1).addActionListener(al);
        submenu2.add(new JMenuItem("Pozycja 11"));
        submenu2.getItem(2).addActionListener(al);
        menu.add(submenu1);
        submenu1.add(submenu2);
        mb.add(menu);
        setJMenuBar(mb);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Tworzymy trzy różne menu (obiekty klasy Menu) o nazwach menu, submenu1 i submenu2. Pierwsze menu otrzymuje etykietę Menu 1, drugie — Pozycja 4, a trzecie — Pozycja 8. Dlaczego akurat takie nazwy etykiet? Otóż każde z menu otrzymuje po trzy pozycje, a następnie do pierwszego dodajemy submenu1, a do niego dodajemy submenu2. Etykieta menu dodawanego staje się ostatnią pozycją menu, do którego zostało ono dodane. Jeśli więc etykietą submenu1 jest Pozycja 4, to po dodaniu submenu1 do menu etykieta Pozycja 4 staje się ostatnią pozycją menu. Jeśli opis nadal nie jest do końca jasny, spójrzmy na rysunek 8.21, który powinien rozwiązać wszelkie wątpliwości.

Obsługą zdarzeń zajmuje się, podobnie jak we wcześniejszych przykładach, obiekt klasy anonimowej implementującej interfejs ActionListener reprezentowany przez prywatne pole al klasy Aplikacja. W związku z tym w stosunku do każdego obiektu reprezentującego daną

pozycję menu jest wywoływana metoda `addActionListener` w postaci `addActionListener(al)`. Wykorzystywany jest tu jednak inny sposób dostępu do tych obiektów. Ponieważ są one tworzone bezpośrednio w metodzie `add` klasy `JMenu`, np.: `menu.add(new JMenuItem("Pozycja 1"))`; to aby otrzymać odpowiednią referencję, wywołujemy metodę `getItem`, podając jako argument indeks wybranego menu. Indeks pierwszego menu ma wartość 0, drugiego — 1 itd.

Nowe instrukcje pojawiły się również w metodzie `actionPerformed`. Jej zadaniem jest wyświetlenie na konsoli nazwy menu, które zostało wybrane przez użytkownika. W celu pobrania tej nazwy jest wykonywana złożona instrukcja:

```
String text = ((JMenuItem)e.getSource()).getText();
```

Obiekt `e` to obiekt klasy `ActionEvent` zawierający wszystkie informacje o zdarzeniu. Metoda `getSource` pozwala na pobranie obiektu, który zapoczątkował dane zdarzenie, a więc obiektu menu wybranego przez użytkownika. Ponieważ typem wartości zwracanej przez `getSource` jest `Object`, dokonujemy rzutowania na typ `JMenuItem`, a potem wywołujemy metodę `getText`, która zwraca nazwę wybranego menu. Uzyskany tekst jest wyświetlany na konsoli za pomocą instrukcji:

```
System.out.println(text);
```

Zwróćmy w tym miejscu uwagę, że taki sposób obsługi był możliwy, jako że jedynymi źródłami zdarzeń były obiekty związane z pozycjami menu, czyli klasy `JMenuItem`.

Zadanie

Jako ćwiczenie do samodzielnego przemyślenia można zaproponować wykonanie takiego samego zadania w sytuacji, kiedy źródłami zdarzeń są również inne komponenty niż `JMenuItem`.

CheckBoxMenu

Oprócz zwykłych menu zaprezentowanych na wcześniejszych stronach pakiet `javax.swing` oferuje też menu, które umożliwiają zaznaczanie poszczególnych pozycji (ang. `checkbox menu`). Za ich reprezentację odpowiada klasa o nazwie `JCheckBoxMenuItem`. Tworzenie tego typu menu odbywa się na takiej samej zasadzie jak zwykłych, z tą różnicą, że zamiast klasy `JMenuItem` korzystamy z `JCheckBoxMenuItem`. Zmienić niestety musimy jednak również sposób obsługi zdarzeń. Wykorzystać należy dodatkowy interfejs o nazwie `ItemListener`. Definiuje on tylko jedną metodę o nazwie `itemStateChanged`, która jest wywoływana za każdym razem, kiedy zmieni się stan komponentu (w naszym przypadku stan pozycji menu). Kod aplikacji zawierającej przykładowe menu posiadające możliwość zaznaczania poszczególnych pozycji został przedstawiony na listingu.

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

public
class Aplikacja extends JFrame
{
    private JCheckBoxMenuItem menuItem1, menuItem2;
    private ItemListener il = new ItemListener()
    {
        public void itemStateChanged(ItemEvent e)
        {
            if(e.getSource() == menuItem1)
            {
```

```

        if(menuItem1.getState())
            System.out.println("Pozycja 1 jest zaznaczona.");
        else
            System.out.println("Pozycja 1 nie jest zaznaczona.");
    }
    else if(e.getSource() == menuItem2)
    {
        if(menuItem2.getState())
            System.out.println("Pozycja 2 jest zaznaczona.");
        else
            System.out.println("Pozycja 2 nie jest zaznaczona.");
    }
}
};
public Aplikacja()
{
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JMenuBar mb = new JMenuBar();
    JMenu menu = new JMenu("Menu 1");
    menuItem1 = new JCheckBoxMenuItem("Pozycja 1", true);
    menuItem1.addItemListener(il);
    menu.add(menuItem1);
    menuItem2 = new JCheckBoxMenuItem("Pozycja 2", false);
    menuItem2.addItemListener(il);
    menu.add(menuItem2);
    mb.add(menu);
    setJMenuBar(mb);
    setSize( 20, 200);
    setVisible(true);
}
public static void main(String args[])
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new Aplikacja();
        }
    });
}
}

```

Struktura menu jest tworzona w taki sam sposób, jak w przypadku elementów typu `MenuItem`. Wykorzystujemy jedynie dodatkowy argument konstruktora klasy `JCheckBoxMenuItem`, który określa, czy dana pozycja ma być zaznaczona (`true`), czy nie (`false`).

Obsługa zdarzeń przychodzących z obiektów klasy `JCheckBoxMenuItem` wymaga implementacji interfejsu `ItemListener`, a tym samym metody `ItemStateChanged`. Metoda ta

będzie wywoływana za każdym razem, kiedy nastąpi zmiana stanu danej pozycji menu, czyli kiedy zostanie ona zaznaczona lub jej zaznaczenie zostanie usunięte. Sprawdzamy wtedy, która pozycja spowodowała wygenerowanie zdarzenia: `penultimate1` czy `penultimate2`. Referencję uzyskujemy przez wywołanie metody `getSource`. Jeśli chcemy sprawdzić, czy dana pozycja jest zaznaczona, czy nie, korzystamy z kolei z metody `getState`. Jeżeli zwróci ona wartość `true`, pozycja jest zaznaczona, jeśli `false` — nie jest. Menu kontekstowe

Jeśli zachodzi potrzeba wyposażenia aplikacji w menu kontekstowe, istnieje oczywiście taka możliwość. Należy wtedy skorzystać z klasy `JPopupMenu`. Konstrukcja taka jest bardzo podobna do zwyczajnego menu, z tą różnicą, że menu kontekstowego nie dodaje się do paska menu.

Przykładowa aplikacja zawierająca menu kontekstowe została przedstawiona

```
import javax.swing.*;
import java.awt.event.*;
public
class Aplikacja extends JFrame
{
    private JPopupMenu popupMenu;
    private JMenuItem miPozycja1, miPozycja2, miZamknij;
    private ActionListener al = new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource() == miZamknij)
                dispose();
        }
    };
    private MouseAdapter ma = new MouseAdapter()
    {
        public void mousePressed(MouseEvent e)
        {
            if(e.isPopupTrigger())
                popupMenu.show(e.getComponent(), e.getX(), e.getY());
        }
        public void mouseReleased(MouseEvent e)
        {
            if(e.isPopupTrigger())
                popupMenu.show(e.getComponent(), e.getX(), e.getY());
        }
    };
    public Aplikacja()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        popupMenu = new JPopupMenu();
        miPozycja1 = new JMenuItem("Pozycja 1");
        miPozycja2 = new JMenuItem("Pozycja 2");
        miZamknij = new JMenuItem("Zamknij");
        miPozycja1.addActionListener(al);
```

```

        miPozycja2.addActionListener(al);
        miZamknij.addActionListener(al);
        popupMenu.add(miPozycja1);
        popupMenu.add(miPozycja2);
        popupMenu.addSeparator();
        popupMenu.add(miZamknij);
        addMouseListener(ma);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Obiekt menu kontekstowego tworzymy, wywołując konstruktor klasy JPopupMenu. Konstruktor ten może być bezargumentowy, tak jak na powyższym listingu, lub też przyjmować jeden argument klasy String. W tym drugim przypadku menu otrzyma identyfikującą je nazwę. Struktura menu jest tworzona w identyczny sposób, jak w przypadku wcześniej omawianych zwyczajnych menu — dodajemy po prostu kolejne obiekty klasy JMenuItem, które staną się pozycjami menu. Dodatkowo za pomocą wywołania metody addSeparator dodany został również separator pozycji menu. Również obsługa zdarzeń jest analogiczna, jak w poprzednich przykładach. Obiektem obsługującym zdarzenia powiązany z każdym z obiektów JMenuItem jest obiekt klasy anonimowej pochodnej od ActionListener. W metodzie actionPerformed sprawdzane jest, czy wybrana pozycja menu to Zapknij (czyli czy obiektem źródłowym zdarzenia jest piZapknij). Jeśli tak, wywoływana jest metoda dispose obiektu aplikacji, która zwalnia zasoby związane z oknem i zamyka okno. Ponieważ jest to jedyne okno aplikacji, czynność ta jest równoznaczna z zakończeniem pracy całego programu.

Aby jednak menu kontekstowe pojawiło się na ekranie, użytkownik aplikacji musi nacisnąć prawy klawisz myszy. To oznacza, że aplikacja musi reagować na zdarzenia związane z obsługą myszy, wymaga to implementacji interfejsu MouseListener. Osiągamy to przez zastosowanie obiektu klasy anonimowej pochodnej od MouseAdapter. Obsługujemy metody mousePressed oraz mouseReleased. Jest to niezbędne, gdyż w różnych systemach w różny sposób wywoływane jest menu kontekstowe. W obu przypadkach za pomocą metody isPopupTrigger sprawdzamy, czy zdarzenie jest wynikiem wywołania przez użytkownika menu kontekstowego. Jeśli tak (wywołanie isPopupTrigger zwróciło wartość true), wyświetlamy menu w miejscu wskazywanym przez kursor myszy. Wyświetlenie menu odbywa się za pomocą metody show obiektu popupMenu. Pierwszym parametrem jest obiekt, na którego powierzchni ma się pojawić menu i względem którego będą obliczane współrzędne wyświetlania, przekazywane jako drugi i trzeci argument.

W naszym przypadku pierwszym argumentem jest po prostu obiekt aplikacji uzyskiwany przez wywołanie `e.getOpponent()`. Metoda `getOpponent` klasy `MouseEvent` zwraca bowiem obiekt, który zapoczątkował zdarzenie.

Zadania

Ćwiczenia do samodzielnego wykonania

Ćwiczenie 41.1.

Popraw kod z listingu 8.26 w taki sposób, aby wyświetlany tekst znajdował się w centrum okna aplikacji.

Ćwiczenie 41.2.

Zmień kod z listingu 8.27 tak, aby obsługa zdarzeń odbywała się poprzez obiekt klasy wewnętrznej implementującej interfejs `WindowListener`.

Lekcja 42. Komponenty 373

Ćwiczenie 41.3.

Zmodyfikuj kod z listingu 8.27 tak, aby obsługa zdarzeń odbywała się poprzez obiekt niezależnej klasy pakietowej implementującej interfejs `WindowListener`.

Ćwiczenie 41.4.

Napisz aplikację zawierającą wielopoziomowe menu kontekstowe.

Ćwiczenie 41.5.

Napisz aplikację, która będzie wyświetlała w swoim oknie aktualne współrzędne kursora myszy.

Ćwiczenie 41.6.

Napisz aplikację zawierającą menu. Po wybraniu z niego dowolnej pozycji powinno pojawić się nowe okno (w tym celu utwórz nowy obiekt klasy `JFrame`). Okno to musi dać się zamknąć przez kliknięcie odpowiedniej ikony z paska tytułu.

Przyciski ciąg dalszy

Obsługą i wyświetlaniem przycisków zajmuje się klasa `JButton`. Podobnie jak w przypadku klasy `JLabel`, konstruktor może być bezargumentowy, powstaje wtedy przycisk bez napisu na jego powierzchni, jak również może przyjmować argument klasy `String`.

W tym drugim przypadku przekazany napis pojawi się na przycisku. Jeśli zastosujemy konstruktor bezargumentowy, będzie możliwe późniejsze przypisanie tekstu przyciskowi za pomocą metody `setText`. W odróżnieniu od etykiet, przyciski powinny jednak reagować na kliknięcia myszą, przy ich stosowaniu niezbędne będzie zatem zaimplementowanie interfejsu `ActionListener`.

Przykładowa aplikacja zawierająca dwa przyciski, taka że po kliknięciu drugiego z nich nastąpi jej zamknięcie,

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

public
class Aplikacja extends JFrame
{
    private JButton button1, button2;
    public Aplikacja()
```



```

{
    ActionListener al = new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource() == button2)
                dispose();
        }
    };
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(null);
    button1 = new JButton("Pierwszy przycisk");
    button1.setBounds(100, 40, 160, 20);
    button1.addActionListener(al);
    button2 = new JButton();
    button2.setText("Drugi przycisk");
    button2.setBounds(100, 80, 160, 20);
    button2.addActionListener(al);
    add(button1);
    add(button2);
    setSize( 20, 200);
    setVisible(true);
}

public static void main(String args[])
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new Aplikacja();
        }
    });
}
}

```

Opis

Pierwszy przycisk jest tworzony za pomocą konstruktora przyjmującego w argumencie obiekt klasy String, czyli po prostu ciąg znaków. W drugim przypadku do ustawiania napisu wyświetlanego na przycisku wykorzystujemy metodę `setText`. Do ustalenia położenia i rozmiarów przycisków wykorzystujemy natomiast metodę `setBounds`, której działanie jest identyczne, jak w przypadku przedstawionych wcześniej etykiet. Ponieważ przyciski muszą reagować na kliknięcia, tworzymy nowy obiekt (al) anonimowej klasy implementującej interfejs `ActionListener`. W metodzie `actionPerformed` sprawdzamy, czy źródłem zdarzenia był drugi przycisk (`if(e.getSource() == button2)`). Jeśli tak, zamykamy okno i kończymy działanie aplikacji (wywołanie metody `dispose`).

Przykład

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

public
class Aplikacja extends JFrame
{
    private JTextField textField;
    private JButton button1;
    public Aplikacja()
    {
        ActionListener al = new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                if(e.getSource() == button1)
                    setTitle(textField.getText());
            }
        };
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        textField = new JTextField();
        textField.setBounds(100, 50, 100, 20);
        button1 = new JButton("Kliknij!");
        button1.setBounds(100, 80, 100, 20);
        button1.addActionListener(al);
        add(button1);
        add(textField);
        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}
```

Pole tekstowe ma szerokość ustawiamy na 100 pikseli, a wysokość na 20. Do dyspozycji mamy również przycisk, którego kliknięcie będzie powodowało, że tekst znajdujący się w polu stanie się tytułem okna aplikacji. Do obsługi zdarzeń wykorzystujemy interfejs

ActionListener i metodę actionPerformed. Tekst zapisany w polu tekstowym odczytujemy za pomocą metody getText, natomiast tytuł okna aplikacji zmieniamy za pomocą metody setTitle.

Jeśli chcemy mieć możliwość reagowania na każdą zmianę tekstu, jaka zachodzi w polu tekstowym JTextField, sytuacja nieco się komplikuje. Otóż należy monitorować zmiany dokumentu (obiektu klasy Document) powiązanego z komponentem JTextField. Trzeba zatem skorzystać z interfejsu DocumentListener, tworząc nowy obiekt implementujący ten interfejs, i przekazać go jako argument metody setDocumentListener wywołanej na rzecz obiektu zwróconego przez wywołanie getDocument klasy JTextField. Zakładając więc, że obiektem implementującym interfejs DocumentListener jest dl, a pole tekstowe reprezentuje obiekt textField, wywołanie powinno mieć postać:

```
textField.getDocument().addDocumentListener(dl)
```

W interfejsie DocumentListener zdefiniowane zostały natomiast trzy metody

- changedUpdate → wywoływana po zmianie atrybutu lub atrybutów;
- insertUpdate → wywoływana przy wstawianiu treści do dokumentu;
- removeUpdate → wywoływana przy usuwaniu treści z dokumentu.

Przykład

Każda zmiana w tym polu powodowałaby zmianę napisu znajdującego się na pasku tytułowym okna,

```
import javax.swing.*;
import javax.swing.event.*;
public
class Aplikacja extends JFrame
{
    private JTextField textField;
    public Aplikacja() {
        DocumentListener dl = new DocumentListener()
        {
            public void changedUpdate(DocumentEvent e)
            {
                setTitle(textField.getText());
            }
        }
        public void insertUpdate(DocumentEvent e)
        {
            setTitle(textField.getText());
        }
        public void removeUpdate(DocumentEvent e)
        {
            setTitle(textField.getText());
        }
    };
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(null);
    textField = new JTextField();
    textField.setBounds(100, 50, 100, 20);
    textField.getDocument().addDocumentListener(dl);
    add(textField);
}
```

```

        setSize( 20, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Zadania

Ćwiczenia do samodzielnego wykonania

Ćwiczenie 42.1.

Napisz aplikację zawierającą pole tekstowe, etykietę i przycisk. Po kliknięciu przycisku zawartość pola tekstowego powinna się znaleźć na etykiecie.

Ćwiczenie 42.2.

Zmodyfikuj kod z listingu 8.38 tak, aby zamknięcie aplikacji następowało jedynie po kliknięciu przycisków w kolejności: Pierwszy przycisk, Drugi przycisk.

Ćwiczenie 42.3.

Do aplikacji z listingu 8.43 dodaj przycisk. Po jego kliknięciu powinny zostać usunięte wszystkie elementy listy.

Obsługa Zdarzeń

Zdarzenia

Aplety są sterowane zdarzeniami.

Większość zdarzeń jest generowana przez:

- mysz
- klawiaturę
- elementy interfejsu graficznego

Obsługa zdarzeń jest zawarta w pakiecie java.awt.event.

Model Delegowania Zdarzeń

Delegowanie zdarzeń:

- źródło generuje zdarzenia
- zdarzenia są wysyłane do słuchaczy

Słuchacz musi zarejestrować się u źródła, aby otrzymywać zawiadomienia o zdarzeniach. Słuchacz normalnie czeka na otrzymanie zdarzenia, po czym obsługuje je i natychmiast powraca.

Zdarzenia i Ich Źródła

Zdarzenie jest obiektem, który opisuje zmianę stanu swojego źródła, spowodowaną np.

- przesunięciem myszki
- naciśnięciem klawisza
- wyborem elementu w liście wyboru
- przepełnienie wartości licznika
- zakończenie działania timera

Źródło jest obiektem, który wygenerował zdarzenie.
Może generować więcej niż jeden rodzaj zdarzenia.

Słuchacze Zdarzeń

Obiekt który otrzymuje zawiadomienia o zdarzeniach:

- jest zarejestrowany u źródła by otrzymywać zawiadomienia o danym rodzaju zdarzeń
- musi implementować metody które otrzymują i przetwarzają te zdarzenia

Rejestracja Słuchacza

Rejestracja słuchacza do otrzymywania powiadomień o wydarzeniach typu Type:

```
public void addTypeListener(TypeListener el)
```

Type - to nazwa zdarzenia (np. Key, MouseMotion)

el - to odwołanie do słuchacza zdarzeń

Wypisanie Słuchacza

Wypisanie odwołanie rejestracji gdy słuchacz już nie jest zainteresowany otrzymywaniem powiadomień o zdarzeniach:

```
public void removeTypeListener(TypeListener el)
```

Obsługa Zdarzeń

Słuchacz musi implementować interfejs dla określonego rodzaju zdarzeń.

Zbiór interfejsów zawarty w java.awt.events.

Na przykład interfejs KeyListener posiada metody:

```
void keyPressed(KeyEvent ke)
```

```
void keyReleased(KeyEvent ke)
```

```
void keyTyped(KeyEvent ke)
```

Klasy Zdarzeń

Nadklasą wszystkich zdarzeń jest java.util.EventObject.

Konstruktor zdarzeń (**src** to obiekt generujący):

```
EventObject(Object src)
```

Metody:

```
Object getSource()
```

```
String toString()
```

```
int getID()
```

AWTEvent jest podklasą EventObject.

Typy Zdarzeń AWT

AWTEvent jest nadklasą różnych typów zdarzeń generowanych przez elementy interfejsu graficznego:

- KeyEvent – generowany gdy otrzymane jest wejście z klawiatury
- MouseEvent – generowany gdy mysz jest przesuwana, klikana, naciskana, zwalniana, itp.
- ActionEvent – generowany gdy naciskany jest przycisk, wybierany element menu, itp.
- TextEvent – generowany gdy zmienia się wartość pola tekstowego
- itp.

Zdarzenia Myszy

MouseEvent jest podklasą InputEvent, która jest podklasą AWTEvent.

Jest kilka typów MouseEvent:

- MOUSE_CLICKED - kliknięcie
- MOUSE_DRAGGED - przeciąganie
- MOUSE_ENTERED – wejście do elementu
- MOUSE_EXITED – wyjście z elementu
- MOUSE_MOVED - przesuwanie
- MOUSE_PRESSED – naciskanie
- MOUSE_RELEASED - zwalnianie

Konstruktor Zdarzenia Myszy

MouseEvent(
 Odwołanie do obiektu który wygenerował zdarzenie:
Component src,
 Typ zdarzenia myszy:
int type,
 Czas systemowy kiedy zdarzenie zaszło:
long when,

Konstruktor Zdarzenia Myszy

Które modyfikatory były wciśnięte:
int modifiers,
Współrzędne myszy:
int x, int y,
Liczba kliknięć:
int clicks,
Czy zdarzenie spowodowało pojawienie się menu?
boolean triggersPopup)

Metody Zdarzenia Myszy

- uzyskanie współrzędnych myszy:
int getX()
int getY()
- zmiana współrzędnych zdarzenia:
void translatePoint(int x, int y)
- ilość kliknięć:
int getClickCount()

Zdarzenia Klawiatury

KeyEvent jest generowany dla wejścia z klawiatury.

Trzy typy zdarzeń klawiatury:

- KEY_PRESSED – klawisz jest naciśnięty
- KEY_RELEASED – klawisz jest zwolniony
- KEY_TYPED – znak jest wygenerowany

Konstruktor zdarzenia klawiatury:

KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)

Zdarzenia Klawiatury

Wirtualny kod klawisza:

VK_ENTER
VK_ESCAPE
VK_CANCEL
VK_UP
VK_DOWN
VK_LEFT
VK_RIGHT
VK_SHIFT
VK_ALT

Metody klasy zdarzeń klawiatury:

char getKeyChar()

int getKeyCode()

CHAR_UNDEFINED gdy znak niedostępny.

VK_UNDEFINED gdy kod niedostępny.

Źródła Zdarzeń

Składowe graficznego interfejsu użytkownika które mogą generować zdarzenia:

- Button – generuje ActionEvent gdy przycisk jest naciskany
- List – generuje ActionEvent gdy element listy jest podwójnie klikany, a ItemEvent gdy element jest wybierany lub zwalniany
- Window – generuje WindowEvent gdy okno jest aktywowane, zamykane, otwierany, minimalizowane, przywracane, usuwane

Interfejs KeyListener

Interfejs do implementacji przez słuchacza klawiatury, aby ten mógł obsługiwać KeyEvent.

Metody:

- void keyPressed(KeyEvent ke)
- void keyReleased(KeyEvent ke)
- void keyTyped(KeyEvent ke)

Użytkownik naciska i zwalnia klawisz 'A'. Trzy zdarzenia: naciśnięty, znak, zwolniony.

Użytkownik naciska i zwalnia klawisz 'home'. Dwa zdarzenia: naciśnięty, zwolniony.

Interfejs MouseListener

Interfejs do implementacji przez słuchacza myszy, aby ten mógł obsługiwać MouseEvent.

Metody:

- void mouseClicked(MouseEvent me)
- void mouseEntered(MouseEvent me)
- void mouseExited(MouseEvent me)
- void mousePressed(MouseEvent me)
- void mouseReleased(MouseEvent me)

Interfejs MouseMotionListener

Interfejs do implementacji przez słuchacza ruchu myszy, aby ten mógł obsługiwać MouseEvent.

Metody:

- void mouseDragged(MouseEvent me)
- void mouseMoved(MouseEvent me)

Obie są wywoływane wielokrotnie gdy mysz jest ciągnana/przesuwana.

Użycie Modelu Delegacji Zdarzeń

Aplikacja musi:

- implementować interfejs słuchacza tak by otrzymać powiadomienia o danym typie zdarzeń
- zarejestrować słuchacza jako odbiorcę powiadomień

Źródło może generować kilka typów zdarzeń.

Słuchacz może rejestrować odbiór wielu rodzajów zdarzeń, jednak musi implementować wszystkie interfejsy dla ich obsługi.

Przykład: Obsługa Zdarzeń Myszy

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents"
width=300 height=100>
</applet>
*/
```

Aplet, implementuje oba interfejsy zdarzeń myszy:

```
public class MouseEvents extends Applet implements MouseListener, MouseMotionListener
{
```

Przykład: Obsługa Zdarzeń Myszy

Wiadomość do wyświetlenia i współrzędne myszy:

```
String msg = "";
int mouseX = 0;
int mouseY = 0;
```

Aplet rejestruje się jako słuchaczy zdarzeń myszy:

```
public void init()
```



```
{  
addMouseListener(this);  
addMouseMotionListener(this);  
}
```

Przykład: Obsługa Zdarzeń Myszy

Implementacja wszystkich metod w interfejsach `MouseListener` i `MouseMotionListener`.
Gdy myszka naciśnięta, tekst w rogu okna apletu:

```
// Handle mouse clicked.  
public void mouseClicked(MouseEvent me)  
{  
    // save coordinates  
    mouseX = 0;  
    mouseY = 10;  
    msg = "Mouse clicked.";   
    repaint();  
}
```

Przykład: Obsługa Zdarzeń Myszy

Gdy wchodzi do (wychodzi z) obszar apletu, wyświetla się tekst w rogu okna apletu:

```
public void mouseEntered(MouseEvent me)  
{  
    mouseX = 0;  
    mouseY = 10;  
    msg = "Mouse entered.";   
    repaint();  
}
```

```
public void mouseExited(MouseEvent me)  
{  
    mouseX = 0;  
    mouseY = 10;  
    msg = "Mouse exited.";   
    repaint();  
}
```

Przykład: Obsługa Zdarzeń Myszy

Gdy myszka przyciśnięta/zwolniona, wyświetla się odpowiedni tekst na bieżącej pozycji:

```
public void mousePressed(MouseEvent me)  
{  
    mouseX = me.getX();  
    mouseY = me.getY();  
    msg = "Down";  
    repaint();  
}
```

```
public void mouseReleased(MouseEvent me)
```

```

{
mouseX = me.getX();
mouseY = me.getY();
msg = "Up";
repaint();
}

```

Przykład: Obsługa Zdarzeń Myszy

Gdy myszka jest ciągniona wyświetla się '*' na bieżącej pozycji i tekst w oknie statusu:

```

public void mouseDragged(MouseEvent me)
{
mouseX = me.getX();
mouseY = me.getY();
msg = "*";
setStatus("Dragging mouse at " +
mouseX + ", " + mouseY);
repaint();
}

```

```

public void mouseMoved(MouseEvent me)
{
setStatus("Moving mouse at " +
me.getX() + ", " + me.getY());
}

```

Przykład: Obsługa Zdarzeń Myszy

Wyświetla się wiadomość na danych współrzędnych:

```

// Display msg in applet window
// at current X,Y location.
public void paint(Graphics g)
{
g.drawString(msg, mouseX, mouseY);
}

```

Obsługa Zdarzeń Klawiatury

Obsługa zdarzeń klawiatury: aplet implementuje interfejs KeyListener.

Za każdym razem, gdy użytkownik naciska klawisz, generowanych jest 2-3 zdarzeń.

Gdy chodzi nam tylko o wpisywane znaki, możemy ignorować zdarzenia naciśnięcia/zwolnienia klawiszy.

Przykład: Obsługa Zdarzeń Klawiatury

Aplet, implementuje interfejs nasłuchu klawiatury:

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="KeyEvents" width=300 height=100>

```

```

</applet>
*/
public class KeyEvents extends Applet
implements KeyListener {
String msg = "";
int X = 10, Y = 20; // output coordinates

```

Przykład: Obsługa Zdarzeń Klawiatury
 Rejestracja apletu jako słuchacza zdarzeń klawiatury:

```

public void init()
{
addKeyListener(this);
requestFocus(); // request input focus
}

```

Obsługa zdarzenia naciśnięcia klawisza:

```

public void keyPressed(KeyEvent ke)
{
showStatus("Key Down");
int key = ke.getKeyCode();
switch(key)
{
//Przykład: Obsługa Zdarzeń Klawiatury
case KeyEvent.VK_F1:
msg += "<F1>";break;
case KeyEvent.VK_F2:
msg += "<F2>";break;
case KeyEvent.VK_F3:
msg += "<F3>";break;
case KeyEvent.VK_PAGE_DOWN:
msg += "<PgDn>";break;
case KeyEvent.VK_PAGE_UP:
msg += "<PgUp>";break;
case KeyEvent.VK_LEFT:
msg += "<Left Arrow>";break;
case KeyEvent.VK_RIGHT:
msg += "<Right Arrow>";break;
}
repaint();
}

```

Przykład: Obsługa Zdarzeń Klawiatury
 Obsługa zdarzenia zwolnienia klawisza:

```

public void keyReleased(KeyEvent ke)
{
showStatus("Key Up");
}

```

```
//Obsługa zdarzenia otrzymania znaku:
public void keyTyped(KeyEvent ke)
{
    msg += ke.getKeyChar();
    repaint();
}
```

Przykład: Obsługa Zdarzeń Klawiatury
Wyświetla komunikat na danych współrzędnych:

```
// Display keystrokes.
public void paint(Graphics g)
{
    g.drawString(msg, X, Y);
}
}
```

Adapter Obsługi Zdarzeń

Adapter: ułatwia tworzenie klas obsługi zdarzeń.

Dostarcza puste implementacje wszystkich metod w danym interfejsie zdarzeń.

Wystarczy dziedziczyć po klasie odpowiedniego adaptera, implementując tylko te zdarzenia które nas interesują.

Przykład: Adapter
Applet, rejestracja słuchacza zdarzeń myszy:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="AdapterDemo"
width=300 height=100>
</applet> */
public class AdapterDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter(this));
        addMouseMotionListener(
            new MyMouseMotionAdapter(this));
    }
}
```

Przykład: Adapter
Klasa obsługująca zdarzenia myszy, implementujetylko jedną metodę:

```
class MyMouseAdapter extends MouseAdapter
{
    AdapterDemo adapterDemo;
    public MyMouseAdapter(
        AdapterDemo adapterDemo){
```

```

this.adapterDemo = adapterDemo;
}
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
    adapterDemo.showStatus("Mouse clicked");
}
}

```

Przykład: Adapter

Klasa obsługująca zdarzenia ruchu myszy, implementuje tylko jedną metodę:

```

class MyMouseMotionAdapter extends MouseMotionAdapter
{
    AdapterDemo adapterDemo;
    public MyMouseMotionAdapter(
        AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }
    // Handle mouse dragged.
    public void mouseDragged(MouseEvent me)
    {
        adapterDemo.showStatus("Mouse dragged");
    }
}

```

Procesy obsługujące zdarzenia -pakiet java.awt.event (1)

- ActionListener - obsługa zdarzeń akcji generowanych przez Użytkownika (kliknięcie przycisku)
- AdjustmentListener - obsługa zdarzeń regulacji generowanych w momencie zmian kontrolerek ekranowych (suwaków, pasków przewijania)
- MouseListener - obsługa zdarzeń generowanych w momencie naciśnięcia przycisków myszki
- WindowListener - obsługa zdarzeń generowanych przez okno (maksymalizacja, minimalizacja, przesunięcie, zamknięcie)
- FocusListener - obsługuje zdarzenia generowane w momencie aktywacji lub dezaktywacji komponentów (np. pole tekstowe staje się lub przestaje być aktywne)
- ItemListener - obsługuje zdarzenia generowane przy zaznaczaniu pola wyboru
- KeyListener - obsługuje zdarzenia generowane przy wpisywaniu tekstu z klawiatury
- MouseMotionListener - obsługuje zdarzenia generowane w momencie przesuwania wskaźnikamyszki nad danym składnikiem

Przygotowanie składników do obsługi zdarzeń (1)

- addActionListener() - dla składników np. Button, CheckBox, TextField (z pakietu AWT) i JButton, JCheckBox, JComboBox, JPasswordField, JRadioButton (z pakietu Swing)
- addAdjustmentListener() - dla ScrollBar, JScrollBar
- addFocusListener() - dla wszystkich składników z pakietu Swing
- addItemListener() - dla JButton, JCheckBox, JComboBox, JRadioButton

- addKeyListener(), addMouseListener(), addMouseMotionListener() - dla wszystkich składników z pakietu Swing
- addWindowListener() - dla obiektów typu Frame i Window (z AWT) oraz JFrame, JWindow (ze Swinga)
- Procesy obsługujące zdarzenia powinny być powiązane z odpowiednimi składnikami i skonfigurowane zanim zostaną dodane do kontenera (w przeciwnym razie wszystkie ustawienia zostaną pominięte)
- Wszystkie metody add... pobierają tylko jeden argument - obiekt, który nasłuchuje zdarzeń danego rodzaju. Zastosowanie słowa kluczowego **this** wskazuje, że zarządcą zdarzeń jest bieżąca klasa

np.

```
Button koniec=new Button("Koniec");
koniec.addActionListener(this);
```

Zdarzenia typu Action (1)

- Aby obsługiwać zdarzenia typu Action, dana klasa musi posiadać zaimplementowany interfejs ActionListener(); metoda addActionListener() musi być wywołana na rzecz każdego składnika generującego zdarzenie typu Action;
- Interfejs ActionListener posiada jedynie metodę actionPerformed(ActionEvent) o prototypie:

```
public void actionPerformed(ActionEvent evt)
{ // kod obsługujący zdarzenie }
```

Zdarzenia typu Action (2)

- Metody getSource() i getActionCommand() wywołują na rzecz obiektu ActionEvent przekazują informację, który obiekt wygenerował zdarzenie; domyślnie polecenie ActionCommand jest opisane za pomocą tekstu powiązanego z danym składnikiem (np. etykieta na przycisku); wywołanie metody setActionCommand(String) zmienia domyślne polecenie dla danego składnika;

np.

```
Button rysuj=new Button("Rysuj");
TextField tekst=new TextField();
rysuj.setActionCommand("Rysowanie");
tekst.setActionCommand("Rysowanie");
```

przykład 1

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class akcja extends Applet implements ActionListener
{ Button b1;
  TextField t1;
  public void paint()
  { b1=new Button("Powiedz cześć!");
    add(b1);
    b1.addActionListener(this);
    t1=new TextField("",8);
    add(t1);
  } //koniec metody paint
  public void actionPerformed(ActionEvent e)
```

```

{ Object ob1=e.getSource();
if (ob1==b1) t1.setText("Cześć!");
} //koniec metody actionPerformed
} //koniec klasy akcja 10

```

Zdarzenia typu Adjustment

- Interfejs AdjustmentListener posiada metodę o prototypie:

```

public void adjustmentValueChanged (AdjustmentEvent evt)
{ // kod obsługujący zdarzenie}

```

- Metoda getValue() zwraca liczbę całkowitą reprezentującą bieżącą wartość na suwaku

przykład 2 Obsługa zdarzeń regulacji

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class regulacja extends Applet implements AdjustmentListener
{ Scrollbar sb1; TextField t1; Label l1;
public void init()
{ setLayout(null);
l1=new Label("Wynik:", Label.RIGHT);
l1.setBounds(36,36,60,26);
add(l1);
t1=new TextField("",8);
t1.setEditable(false);
t1.setBounds(108,36,48,26);
add(t1);
sb1=new Scrollbar(Scrollbar.HORIZONTAL,0,10,0,100);
sb1.setBounds(180,36,137,26); add(sb1);
t1.setText(String.valueOf(sb1.getValue()));
sb1.addAdjustmentListener(this); } 12

```

```

public void adjustmentValueChanged(AdjustmentEvent e)
{ Object ob1=e.getSource();
if (ob1==sb1)
{ int wynik=sb1.getValue();
t1.setText(String.valueOf(wynik));
}
}
}

```

Zdarzenia typu Window:

- otwarcie lub zamknięcie okna (np. obiektu Frame,JFrame);

Interfejs WindowListener :

- WindowActivated(WindowEvent)
- WindowClosed(WindowEvent)
- WindowClosing(WindowEvent)
- WindowDeactivated(WindowEvent)

- WindowDeiconified(WindowEvent)
- WindowIconified(WindowEvent)
- WindowOpened(WindowEvent)

przykład 3-Obsługa zdarzeń okna

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;

public class okno extends Object implements WindowListener
{ public void windowOpened(WindowEvent e)
{ System.out.println("Okno otwarte.");
}
public void windowClosing(WindowEvent e)
{ System.out.println("Okno zamyka sie.");System.exit(0);
}
public void windowClosed(WindowEvent e)
{ System.out.println("Okno zamkniete.");
}
public void windowIconified(WindowEvent e)
{ System.out.println("Okno zikonizowane.");
}
public void windowDeiconified(WindowEvent e)
{ System.out.println("Okno przywroczone.");
}
public void windowActivated(WindowEvent e)
{ System.out.println("Okno aktywne.");
}
public void windowDeactivated(WindowEvent e)
{ System.out.println("Okno nieaktywne.");
}
public static void main(String arg [ ] )
{ Frame f=new Frame("Okna");
f.addWindowListener(new okno());
f.setTitle("Obsługa zdarzeń okna");
f.add(new Label("Powodzenia!!!"),"Center");
f.pack(); f.setVisible(true);
} }
```

Zdarzenia typu Mouse (1)

Zdarzenia typu Mouse:

- naciśnięcie dowolnego przycisku myszki
- ustawienie wskaźnika myszki nad obszarem zajmowanym przez dany składnik
- Upuszczenie przez wskaźnik myszki obszaru zajmowanego przez dany składnik

Interfejs MouseListener:

- mouseClicked(MouseEvent)
- mouseEntered(MouseEvent)

- mouseExited(MouseEvent)
- mousePressed(MouseEvent)
- mouseReleased(MouseEvent)

Metody wywoływane na rzecz obiektów

MouseEvent

- getClickCount() - zwraca liczbę kliknięć przyciskiem myszy
- getPoint() - zwraca obiekt Point reprezentujący współrzędne (x,y) miejsca w obszarze zajmowanym przez składnik, w którym został naciśnięty przycisk myszki
- getX() - zwraca współrzędną X powyższej pozycji
- getY() - zwraca współrzędną Y

przykład 4-Obsługa zdarzeń myszy -

```
import java.awt.*;
import java.awt.event.*;
class mysz extends Frame implements MouseListener, ActionListener
{ public static void main(String args[])
{ new mysz(); }
mysz()
{ setLayout(new FlowLayout());
Button b1=new Button("A | 1");
Button b2=new Button("A | 2");
Button b3=new Button("A | 3");
b1.addActionListener(this); b1.addMouseListener(this);
b2.addActionListener(this); b2.addMouseListener(this);
b3.addActionListener(this); b3.addMouseListener(this);
add(b1); add(b2); add(b3);
pack();
setVisible(true);
}
public void mouseEntered(MouseEvent e)
{ e.getComponent().setBackground(Color.yellow);
}
public void mouseExited(MouseEvent e)
{ e.getComponent().setBackground(SystemColor.control);
}
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseClicked(MouseEvent e) { }
public void actionPerformed(ActionEvent e)
{ System.out.println("Akcja wykonana na "+e.getActionCommand());
}
}
```

Adaptery

- Adaptery - puste implementacje interfejsów nasłuchu (np. klasa MouseAdapter definiuje puste metody: mousePressed, mouseReleased,

mouseEntered,
mouseExited,
mouseClicked)

- dostępne w pakiecie java.awt.event

- Sposoby wykorzystania:

- jeśli klasa nie dziedziczy innych i jeśli obsługuje tylko jeden rodzaj zdarzeń - adapter

można odziedziczyć i przeddefiniować jego wybrane metody

- wykorzystanie anonimowych klas wewnętrznych (klasa wewnętrzna - zdefiniowana w innej klasie; anonimowe klasy wewnętrzne - nie mają nazwy a ich definicja pojawia się w instrukcji new)

Dziedziczenie adapteru

```
class X extends MouseAdapter
```

```
{  
public void mouseEntered(MouseEvent e)  
{  
//przedefiniowanie metody mouseEntered  
}  
}
```

Anonimowe klasy wewnętrzne

- Utworzenie obiektu - słuchacza zdarzeń:

np.

```
MouseAdapter ma=new MouseAdapter()  
{  
public void mouseEntered(MouseEvent e)  
{  
// obsługa zdarzenia  
}  
};
```

- przyłączenie utworzonego słuchacza do zdarzenia:

```
x.addMouseListener(ma);
```

- powyższe 2 konstrukcje można zapisać w jednym:

np.

```
x.addMouseListener(new MouseAdapter()  
{  
public void mouseEntered(MouseEvent e)  
{  
// obsługa zdarzenia  
}  
});
```

Przykład 4a – zastosowanie anonimowej klasy wewnętrznej

```
import java.awt.*;  
import java.awt.event.*;  
class mysz1 extends Frame implements ActionListener  
{ public static void main(String args[])  
{ new mysz1(); }
```

```

mysz1()
{ setLayout(new FlowLayout());
//uzycie anonimowej klasy wewnętrznej
MouseListener m1=new MouseAdapter() {
public void mouseEntered(MouseEvent e)
{ e.getComponent().setBackground(Color.yellow); }
public void mouseExited(MouseEvent e)
{ e.getComponent().setBackground(SystemColor.control); }
}; //koniec klasy wewnętrznej
for (int i=0;i<3;i++){
Button b=new Button("A | "+(i+1));
b.addActionListener(this);
b.addMouseListener(m1); //dodajemy wewnętrznego słuchacza
add(b);
}
pack();
setVisible(true);
}
public void actionPerformed(ActionEvent e)
{ System.out.println("Akcja wykonana na " +e.getActionCommand());
}
}

```

Zdarzenia typu Item

- Zdarzenia na komponentach wyboru (Button, RadioButton, CheckBox, Combobox, Choice, List): zaznaczenie lub usunięcie zaznaczenia

Interfejs ItemListener posiada jedną metodę o prototypie:

- void itemStateChanged(ItemEvent e)

Metody wywoływane na rzecz obiektów ItemEvent:

- getItem() - podaje informację o obiekcie, który wygenerował zdarzenie
- getStateChange() - w wyniku otrzymujemy informację czy dany element został zaznaczony (ItemEvent.SELECTED) lub czy jego zaznaczenie zostało usunięte (ItemEvent.DESELECTED)

przykład 5-Obsługa zdarzeń wyboru

```

import java.awt.*;
import java.awt.event.*;
public class wybor extends Frame implements ItemListener
{
public static void main(String args[])
{ new wybor();
}
wybor()
{ setLayout(new FlowLayout());
List l1=new List();
l1.add("Navigator"); l1.add("Explorer"); l1.add("Mozilla");
l1.addItemListener(this);
add(l1); pack(); show();
}
}

```

```

} //koniec metody wybor
public void itemStateChanged(ItemEvent e)
{
if (e.getStateChange()==ItemEvent.SELECTED)
{ System.out.println("Element "+e.getItem()+
" "+e.getItemSelectable()+"-zaznaczony");}
else
{ System.out.println("Element "+e.getItem()+
" "+e.getItemSelectable()+"-odznaczony");}
} //koniec metody itemStateChanged
} //koniec klasy wybor

```

Nasłuch tekstów - zdarzenie

textValueChanged

- zachodzi gdy tekst w komponencie tekstowym (TextField, TextArea) zmienia się
- do pól tekstowych można przyłączyć słuchacza akcji
- akcja powstaje po naciśnięciu klawisza Enter
- getActionCommand - zwraca domyślnie zawartość pola tekstowego

Przykład 6 – sprawdzanie poprawności tekstu

```

import java.awt.*;
import java.awt.event.*;
class teksty extends Frame {
String t0="";
public static void main(String args[])
{ new teksty();
}
teksty() {
TextField tf=new TextField(40);
tf.addTextListener(new TextListener() { //klasa wewnętrzna
public void textValueChanged(TextEvent e){
TextComponent t=(TextComponent) e.getSource();
try {int i=Integer.parseInt(t.getText()); } //łańcuch znakowy
//przekształcany w wartość typu int
catch (NumberFormatException exc) { //gdy nie całkowita:
getToolkit().beep();
t.setText(t0);
t0=t.getText();
} //koniec bloku catch
} //koniec textValueChanged
} ); //koniec klasy wewnętrznej
add(tf);
pack();
setVisible(true);
} //koniec konstruktora teksty
}

```

Zdarzenie typu focus:

- dowolny składnik GUI staje się lub przestaje być aktywny Interfejs FocusListener:

- focusGained(focusEvent)
- focusLost(focusEvent)

Do sprawdzenia informacji, który obiekt wygenerował zdarzenie służy metoda getSource()

Przykład 7 - zdarzenie typu focus klasa fokus1

```
class fokus1 extends Panel implements FocusListener {
    TextField tf;
    final Color NOTFOCUS=Color.gray, FOCUS=Color.blue,
    ERROR=Color.red;
    Color akt_kolor=FOCUS;
    fokus1(int kol) {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setBackground(NOTFOCUS);
        tf=new TextField(kol);
        tf.addFocusListener(this);
        add(tf);
    } //koniec konstruktora fokus
    public void focusGained(FocusEvent e) {
        setBackground(akt_kolor); }
    public void focusLost(FocusEvent e) {
        String s=tf.getText();
        if(!s.equals("")){
            try{ int i=Integer.parseInt(s);}
            catch (NumberFormatException exc) { //gdy nie całkowita:
                getToolkit().beep();
                akt_kolor=ERROR;
                tf.requestFocus(); //przywrócenie fokusu elementowi tf
            } return; } //koniec bloku catch
        } //koniec if
        akt_kolor=FOCUS;
        setBackground(NOTFOCUS);
    } } //koniec klasy fokus1
```

Przykład 7 - aplikacja

```
import java.awt.*;
import java.awt.event.*;
public class test1 extends Frame{
    test1() {
        fokus1 fok1=new fokus1(20), fok2=new fokus1(40);
        setLayout(new GridLayout(2,1));
        add(fok1);
        add(fok2);
        pack();
        setVisible(true);
    }
    public static
    void main(String arg[])
    { new test1(); }
```

}

Zdarzenia typu Key

KeyEvent jest generowany dla wejścia z klawiatury.

Trzy typy zdarzeń klawiatury:

- KEY_PRESSED – klawisz jest naciśnięty
- KEY_RELEASED – klawisz jest zwolniony
- KEY_TYPED – znak jest wygenerowany

Konstruktor zdarzenia klawiatury:

KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)

Zdarzenia Klawiatury

Wirtualny kod klawisza:

VK_ENTER
VK_ESCAPE
VK_CANCEL
VK_UP
VK_DOWN
VK_LEFT
VK_RIGHT
VK_SHIFT
VK_ALT

Metody klasy zdarzeń klawiatury:

char getKeyChar()
int getKeyCode()
CHAR_UNDEFINED gdy znak niedostępny.
VK_UNDEFINED gdy kod niedostępny.

Metoda getKeyChar() wywołana na rzecz obiektu KeyEvent zwraca znak przypisany do klawisza, który wygenerował zdarzenie.

Przykład: Obsługa Zdarzeń Klawiatury

Rejestracja apletu jako słuchacza zdarzeń klawiatury:

```
public void init()
{
    addKeyListener(this);
    requestFocus(); // request input focus
}
```

Obsługa zdarzenia naciśnięcia klawisza:

```
public void keyPressed(KeyEvent ke)
{
    showStatus("Key Down");
    int key = ke.getKeyCode();
    switch(key)
```

```

{
//Przykład: Obsługa Zdarzeń Klawiatury
case KeyEvent.VK_F1:
msg += "<F1>";break;
case KeyEvent.VK_F2:
msg += "<F2>";break;
case KeyEvent.VK_F3:
msg += "<F3>";break;
case KeyEvent.VK_PAGE_DOWN:
msg += "<PgDn>";break;
case KeyEvent.VK_PAGE_UP:
msg += "<PgUp>";break;
case KeyEvent.VK_LEFT:
msg += "<Left Arrow>";break;
case KeyEvent.VK_RIGHT:
msg += "<Right Arrow>";break;
}
repaint();
}

```

Przykład: Obsługa Zdarzeń Klawiatury
Obsługa zdarzenia zwolnienia klawisza:

```

public void keyReleased(KeyEvent ke)
{
showStatus("Key Up");
}

//Obsługa zdarzenia otrzymania znaku:
public void keyTyped(KeyEvent ke)
{
msg += ke.getKeyChar();
repaint();
}

```

Przykład: Obsługa Zdarzeń Klawiatury
Wyświetla komunikat na danych współrzędnych:

```

// Display keystrokes.
public void paint(Graphics g)
{
g.drawString(msg, X, Y);
}
}

```

Dynamiczne przyłączanie i odłączanie słuchaczy

- słuchacza s typu XXX (Action, Mouse, Key, Text itp.) przyłączonego do źródła z można odłączyć za pomocą: `z.removeXXXListener(s);`

- odłączenie słuchacza s (czyli konkretnego obiektu implementującego interfejs nasłuchu) od źródła z oznacza, że ten słuchacz nie będzie obsługiwał zdarzeń dla tego źródła (jeśli są inni słuchacze nadal przyłączeni, to nadal obsługują zdarzenia
- wygodny sposób organizowania różnych działań

Separacja

- Delegacyjny model obsługi zdarzeń w Javie -oddelegowanie obsługi do innego obiektu niż ten, któremu przytrafia się zdarzenie
- koncepcja powyższa pozwala izolować na poziomie merytorycznym różne fragmenty aplikacji, np. można oddzielić kod GUI od kodu odpowiedzialnego za obsługę zdarzeń

Przykład - separacja

- robocza część aplikacji - klasa Robocza (zawiera metody wykonujące konkretne działania np. obliczenia, dodawanie elementów do bazy danych)
- interfejs użytkownika - klasa Gui
- klasa Robocza nie musi „wiedzieć” jaki jest interfejs graficzny; klasa Gui nie musi wiedzieć jakie czynności daje użytkownikowi do wyboru ani jak są obsługiwane
- obiekt klasy Robocza będzie słuchaczem zdarzeń przychodzących z interfejsu

Przykład - klasa Robocza

```
import java.awt.event.*;
public class Robocza implements ActionListener{
    private Gui gui=null;
    public static void main(String args[]) {
        Robocza rob=new Robocza(); }
    public Robocza() {
        //etykiety oznaczające czynności:
        String etykiety[]={ "Dodawanie","Zaznaczanie","Kasowanie","Koniec"};
        //nazwy poleceń związanych z czynnościami:
        String komendy[]={ "dodaj","zaznacz","kasuj","koniec"};
        int krytyczne[]={2,3}; //indeksy czynności krytycznych
        //tworzenie GUI; ostatni argument - słuchacz akcji
        gui=new Gui(etykiety, komendy, krytyczne, this);
        public void actionPerformed(ActionEvent e) {
            String komenda=e.getActionCommand();
            if (komenda.equals("dodaj")) dodaj();
            else if(komenda.equals("zaznacz")) zaznacz();
            else if(komenda.equals("kasuj")) kasuj();
            else if(komenda.equals("koniec")) koniec();
        }
        void koniec() {
            System.exit(0);}
        void dodaj(){ }
        void zaznacz(){ }
        void kasuj(){ }
    }
}
```

Przykład - klasa Gui


```

import java.awt.*;
import java.awt.event.*;
public class Gui extends Frame{
public Gui(String etykiety[], String komendy[], int krytyczne[],
ActionListener a) {
super("Test"); setLayout(new GridLayout());
int n=etykiety.length;
Button b[]=new Button[n];
for(int i=0;i<n;i++){
b[i]=new Button(etykiety[i]);
b[i].setActionCommand(komendy[i]);
b[i].addActionListener(a);
add(b[i]);
}
//tylko krytyczne odpowiedzą słuchaczowi myszy
for (int i=0;i<krytyczne.length;i++)
b[krytyczne[i]].addMouseListener(m1);
pack();
setVisible(true);
}
MouseListener m1=new MouseAdapter(){
public void mouseEntered(MouseEvent e){
e.getComponent().setBackground(Color.orange); }
public void mouseExited(MouseEvent e){
e.getComponent().setBackground(SystemColor.control); }
};
}

```

```
import javax.swing.JApplet;
public class Aplet extends JApplet
{
    public void init( )
    {
        String tek1 = getParameter("tekst1");
        String tek2 = getParameter("tekst2");
        System.out.println(tek1);
        System.out.println(tek2);
    }
}
```

Wy tłumaczenie działania aletu:

W przypadku niektórych apletów istnieje potrzeba przekazania im parametrów z kodu HTML na przykład wartości sterujących ich pracą. Taka możliwość oczywiście istnieje, znacznik <applet> (a także <object>) pozwala na zastosowanie argumentów o nazwie parap.

```
import javax.swing.JApplet;
import java.awt.*;
public class Aplet extends JApplet {
    private String tekst;
    public void init() {
        if((tekst = getParameter("tekst")) == null)
            tekst = "Nie został podany parametr: tekst";
    }
    public void paint (Graphics gDC) {
        gDC.clearRect(0, 0, getSize().width, getSize().height);
        gDC.drawString (tekst, 60, 50);
    }
}
```

Do odczytu wartości parametru wykorzystaliśmy metodę getParapeter, której jako argument należy przekazać nazwę parametru (w naszym przypadku tekst). Metoda ta zwróci wartość wskazanego argumentu lub wartość null, jeżeli parametr nie został uwzględniony w kodzie HTML.

W klasie Aplet deklarujemy zatem pole typu String o nazwie tekst. W metodzie init, która zostanie wykonana po załadowaniu apletu przez przeglądarkę, odczytujemy wartość parametru i przypisujemy ją polu tekst. Sprawdzamy jednocześnie, czy pole to nie otrzymało wartości null — jeśli tak, przypisujemy mu własny tekst. Działanie metody paint jest analogiczne jak we wcześniejszych przykładach, wyświetla ona po prostu zawartość pola tekst na ekranie, w miejscu o wskazanych współrzędnych (x = 60, y = 50).

Przykład obsługi grafiki

```
import javax.swing.JApplet;
```

```

import java.awt.*;
public class Aplet extends JApplet {
    Image img;
    public void init( ) {
        img = getImage(getDocumentBase( ), "image.jpg");
    }
    public void paint(Graphics gDC) {
        gDC.clearRect(0, 0, getSize( ).width, getSize( ).height);
        gDC.drawImage (img, 20, 20, this);
    }
}

```

Obsługa myszy

```

import javax.swing.JApplet;
import java.awt.*;
import java.awt.event.*;
public class Aplet extends JApplet implements MouseListener {
    String tekst = "";
    public void init( ) {
        addMouseListener(this);
    }
    public void paint (Graphics gDC) {
        gDC.setColor(Color.WHITE);
        gDC.clearRect(0, 0, getSize( ).width, getSize( ).height);
        gDC.setColor(Color.BLACK);
        gDC.drawString(tekst, 20, 20);
    }
    public void mouseClicked(MouseEvent evt) {
        int button = evt.getButton( );
        switch(button) {
            case MouseEvent.BUTTON1 : tekst = "Przycisk 1,
";break;
            case MouseEvent.BUTTON2 : tekst = "Przycisk 2,
";break;
            case MouseEvent.BUTTON3 : tekst = "Przycisk 3,
";break;
            default : tekst = "";
        }
        tekst += "współrzędne: x = " + evt.getX( ) + " , ";
        tekst += "y = " + evt.getY( );
        repaint( );
    }
    public void mouseEntered(MouseEvent evt) { }
}

```

```
public void mouseExited(MouseEvent evt) { }  
public void mousePressed(MouseEvent evt) { }  
public void mouseReleased(MouseEvent evt) { }  
}
```

1. Klasa i obiekt

2. Polimorfizm

Funkcje polimorficzne to funkcje nie zwracające uwagi na to, jakiego typu zmienne są wykorzystywane przez nie. W Javie upraszczanie programów przy wykorzystaniu funkcji polimorficznych możliwe jest na trzy sposoby:

dziedziczenie - podklasy automatycznie dziedziczą metody klas nadrzędnych; ponadto każda funkcja, której argumentem może być klasa, zaakceptuje jako argument również każdą podklasę tej klasy;

przeciążenie - możliwe jest użycie w obrębie tej samej klasy metod o identycznych nazwach, ale różniących się typami przyjmowanych argumentów;

interfejs - wykorzystywanie w różnych klasach metod o identycznych nazwach, przyjmujących argumenty tego samego typu, przydatny ze względu na brak wielodziedziczenia w Java

3. Modyfikatory dostęności

Pytania do wyjątków.

Wymień dwie sytuacje specjalne, które mogą wystąpić podczas działania programów. Jak nazywane są sytuacje specjalne oraz z jakiego języka są one zapożyczone.

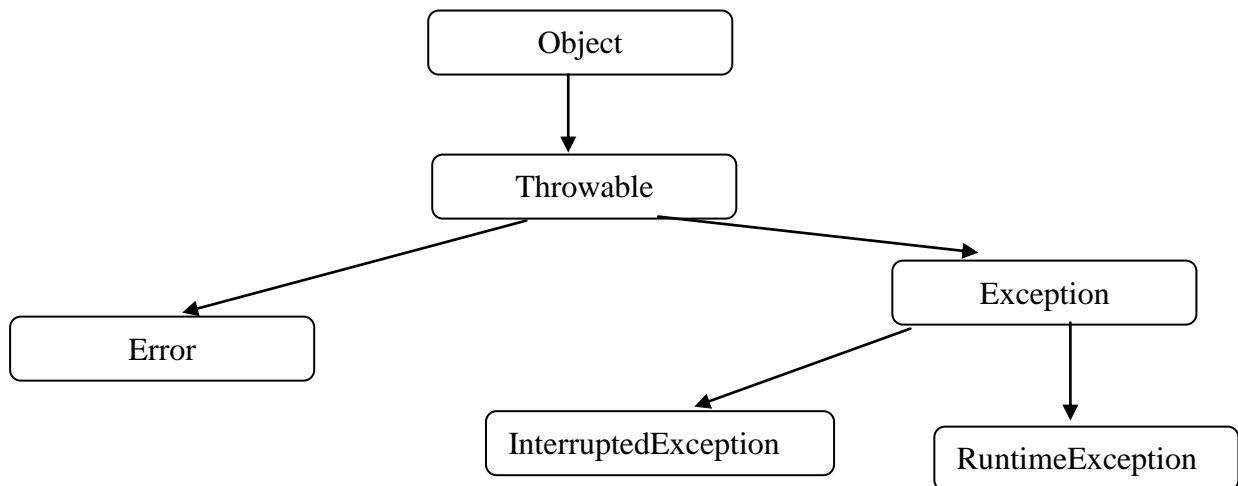
Wyjątki

Podczas działania programu mogą wystąpić różne sytuacje specjalne, do których należą między innymi, wystąpienia błędu polegającego na próbie otwarcia pliku, który nie istnieje lub występuje dzielenie przez zero.

Java posiada zapożyczony z języka Ada mechanizm informowania o błędach: wyjątki (ang. exceptions). Mechanizm obsługi wyjątków w Javie umożliwia zaprogramowanie "wyjścia" z takich sytuacji krytycznych, dzięki czemu program nie zawiesi się po wystąpieniu błędu wykonując ciąg operacji obsługujących wyjątek. Generowanie i obsługę sytuacji wyjątkowych w Javie zrealizowano przy wykorzystaniu paradygmatu programowania zorientowanego obiektowo. Wystąpienie sytuacji wyjątkowej przerywa "normalny" tok wykonywania programu. W Javie sytuacja wyjątkowa występuje wtedy, gdy program wykona instrukcję throw. Wyrażenie throw przekazuje sterowanie do skojarzonego z nim bloku catch (łap, blok obsługujący wystąpienie sytuacji wyjątkowej). Jeśli nie ma bloku catch w bieżącej metodzie, sterowanie natychmiastowo, bez zwracania wartości, przekazywane jest do metody, która wywołała bieżącą funkcję. W tej metodzie szukany jest blok catch. Jeśli blok catch nie zostanie znaleziony, przekazuje sterowanie do metody, która wywołała tę metodę. Sterowanie przekazywane jest zatem zgodnie z łańcuchem wywołań metod, aż do momentu znalezienia bloku catch odpowiedzialnego za obsługę wyjątku.

Wszystkie wyjątki, jakie mogą wystąpić w programie muszą być podklasą klasy Throwable. Poniższy rysunek pokazuje hierarchię dziedziczenia klasy Throwable i jej najważniejszych podklas.

Hierarchia dziedziczenia klas wyjątków



Wyjątki typu Error występują wtedy, gdy wystąpi sytuacja specjalna w maszynie wirtualnej (np. błąd podczas dynamicznego łączenia). Wyjątki tego typu nie powinny być obsługiwane w "zwykłych" programach Javy. Jest także mało prawdopodobne, że typowy program Javy spowoduje wystąpienie wyjątku tego typu. W większości programów generowane są i obsługiwane obiekty, które dziedziczą z klasy Exception. Wyjątek tego typu oznacza, że w programie wystąpił błąd, lecz nie jest to poważny błąd systemowy. Szczególną podklasę klasy Exception stanowią wyjątki, które występują podczas wykonywania programu, są to wyjątki typu RunTimeExceptions (i jej podklas np.: NullPointerException, ClassCastException, IllegalThreadStateException i ArrayOutOfBoundsException) i występują np.: wtedy, gdy zostaną wyczerpane zasoby systemowe, nastąpi odwołanie do nie istniejącego elementu tablicy i inne. Gdy wyjątek taki nie jest obsługowany, program zostaje zatrzymany, a na ekranie pojawia się nazwa wyjątku oraz klasa i metoda, w której wystąpił. Dzięki temu wiemy, w którym miejscu kodu wystąpił błąd i można go szybko poprawić.

Definiowanie i zgłaszanie wyjątków

W języku Java można deklarować nowe klasy wyjątków poprzez rozszerzanie klasy Exception.

Jednym z celów rozszerzania klas wyjątków jest zapewnienie dostarczania przez klasę potomną komunikatów opisujących zaistniałą sytuację awaryjną. Użykuje się to poprzez zdefiniowanie w klasie potomnej odpowiednich pól z tekstami komunikatów, konstruktorów z parametrami napisowymi (komunikatami) oraz poprzez redefinicję metody getMessage() zwracającej napis/komunikat.

Do zgłaszania wyjątków służy instrukcja throw stosowana najczęściej w postaci:

```
throw new KonstruktorKlasyWyjątku(listaParametrów)
```

Java wymaga deklarowania wyjątków kontrolowanych, zgłaszanych przez metodę. Listę wyjątków (nazw klas wyjątków oddzielonych przecinkami) podaje się w nagłówku metody po słowie kluczowym throws.

Jeśli wewnątrz metody wywołujemy metodę, która w swojej klauzuli throws wymienia jakiś wyjątek to mamy trzy wyjścia:

- wychwycić ten wyjątek i go obsłużyć,
- wychwycić ten wyjątek i wygenerować nowy zadeklarowany w klauzuli throws naszej metody (zamiana wyjątku),
- wymienić ten wyjątek w klauzuli throws naszej metody i nie zajmować się jego obsługą; wyjątek jest propagowany dalej i trzeba się będzie nim zająć w miejscu wywołania naszej metody.

Do obsługi wyjątków służy instrukcja try-catch-finally o składni:

```
try
{
    instrukcje
}
catch (typWyjątku identyfikator)
{
    instrukcje
}
catch (typWyjątku identyfikator)
{
    instrukcje
}
...
finally
{
    instrukcje
}
```

Fragment kodu, w którym (potencjalnie) może zostać wyrzucony wyjątek umieszczany jest w bloku try.

Przykład

```
class Wyjatek
{
    public static void main (String args[])
    {
        Punkt punkt = null;
        int liczba;
        try
        {
            liczba = 10 / 0;
            punkt.x = liczba;
        }
    }
}
```

```

    }
    catch(ArithmeticException e)
    {
        System.out.println("Nieprawidłowa operacja arytmetyczna");
        System.out.println(e);
    }
    catch(Exception e)
    {
        System.out.println("Błąd ogólny");
        System.out.println(e);
    }
}

```

Jeśli wyjątek nie wystąpi, to po wykonaniu wszystkich instrukcji bloku try, bloki catch są pomijane.

Instrukcje bloku finally, jeśli został użyty (jest opcjonalny) wykonywane są zawsze, niezależnie od tego, czy w wykonywanym bloku try wyrzucony był wyjątek i nastąpiło przejście do jednego z bloków catch czy też nie.

Pierwsze wystąpienie wyjątku w bloku try powoduje przejście do odpowiedniego bloku catch, którego typ deklarowanego wyjątku jest identyczny z typem wyrzuconego wyjątku lub jest typu ogólniejszego (jest nadklasą).

Klauzula try może mieć wiele klauzul catch, ale muszą one obsługiwać różne typy wyjątków.

Klauzule catch, w celu dopasowania wyjątku, przeszukiwane są kolejno.

Wyjątki typów bardziej szczególnych (podklas) muszą być związane z wcześniejszymi klauzulami catch danej klauzuli try.

Zawsze wykonywana jest co najwyżej jedna klauzula catch.

Jeśli żaden z bloków catch nie wychwyci wyjątku, to jest on propagowany na zewnątrz instrukcji try-catch-finally (i tam powinien być obsługony).

Wyjątki wyrzucone wewnątrz bloków catch i finally również propagowane są na zewnątrz.

Instrukcje try-catch-finally mogą być zagnieżdżane.

program 3.5

```

class Main
{
    public static void main (String args[])
    {
        Punkt punkt = null;
        int liczba;
    }
}

```



```

try{
try{
    liczba = 10 / 0;
}
catch(ArithmeticException e){
    System.out.println("Nieprawidłowa operacja arytmetyczna");
    System.out.println("Przypisuję zmiennejliczba wartość 10");
    liczba = 10;
}
punkt.x = liczba;
}
catch(Exception e){
    System.out.println("Błąd ogólny");
    System.out.println(e);
}
}
}

```

JAVA -Wyjątki i strumienie

Schemat obsługi wyjątków

- wyjątek powstaje na skutek jakiegoś nieoczekiwanego błędu
- wyjątek jest zgłaszany
- wyjątek jest obsługiwany

```

int a,b,c; String s;
try
{
//w bloku try umieszcza się instrukcje,
//które mogą spowodować wyjątek
a=b/c;
//jeżeli c=0 zostanie zgłoszony
//wyjątek ArithmeticException
s=Integer.toString(a);
}
catch (ArithmeticException e)
{
//wyjątek jest obsługiwany w bloku catch
s="*";
}

```

Klasy wyjątków

- wyjątki są obiektami klas wyjątków
- każda nazwa wyjątku - to nazwa klasy
- w hierarchii dziedziczenia - klasa bazowa Throwable
- błędy fatalne (fatal error) są wyprowadzane z podklasy Error klasy Throwable
- pozostałe błędy - podklasy klasy Exception
- w przypadku operacji na liczbach rzeczywistych przydzieleniu przez 0 wyjątek nie jest zgłoszony a wartość wyniku jest postaci POSITIVE_INFINITY (lub NEGATIVE_INFINITY), co po przekształceniu na String da napis: "Infinity" (lub "-Infinity")

Wyjątki kontrolowane i niekontrolowane

- pochodne od klas RuntimeException i Error – wyjątki niekontrolowane
 - pozostałe wyjątki - kontrolowane, tzn.:
 - metody zgłaszające te wyjątki wymieniają je jawnie w swojej deklaracji w klauzuli throws
 - metody te mogą zgłaszać tylko wymienione w klauzuli throws wyjątki lub wyjątki ich podklas
 - odwołania do tych metod wymagają jawnej obsługi ewentualnie zgłaszanych wyjątków:
 - poprzez konstrukcję try-catch lub
 - poprzez wymienienie wyjątku w klauzuli throws
- naszej metody (przesunięcie obsługi wyjątku do miejsca wywołania naszej metody)

Sekwencja działania mechanizmu wyjątków

- wykonywane są kolejne instrukcje bloku try
- jeśli powstaje wyjątek - wykonanie bloku try jest przerywane
- sterowanie przekazywane jest do pierwszej w kolejności klauzuli catch, której typ wyjątku pasuje do typu powstałego wyjątku (należy podawać najpierw te bardziej szczegółowe typy wyjątków)
- inne klauzule catch nie są uruchamiane
- obsługująca wyjątek klauzula catch może np. zmienić sekwencję sterowania (poprzez return lub zgłoszenie nowego wyjątku za pomocą instrukcji throw). Jeśli nie zmienia sekwencji sterowania, to wykonanie programu jest kontynuowane w klauzuli finally (lub jeśli jej nie ma - od następnej instrukcji po try - po ostatniej klauzuli bloku catch)

Klauzula finally

- służy do wykonania kodu niezależnie od tego czy wystąpił wyjątek czy nie
- np.

```
boolean metoda(...) {  
    try { //instrukcje, które mogą spowodować wyjątek  
    }  
    catch (Exception e) { return false; }  
    finally { //uporządkowanie  
    }  
    return true;  
}
```

- jeśli zaistniał wyjątek - uruchamiana jest klauzula catch, która zwraca false na znak niepowodzenia i sterowanie przekazywane jest do klauzuli finally; dopiero potem zwracany jest wynik - false
- jeśli nie było wyjątku - po zakończeniu instrukcji w bloku try sterowanie od razu wchodzi do klauzuli finally

Własne wyjątki

- wyjątki są obiektami klas pochodnych od Throwable, aby stworzyć własny wyjątek należy zdefiniować klasę dziedziczącą podklasę Exception klasy Throwable,

np.:

```
class NowyWyjatek extends Exception
```

```
{  
...  
}
```

w której można zawrzeć pola i metody służące dodatkowej informacji o przyczynach powstania wyjątku

Zgłaszanie wyjątku:

- jeśli nasza metoda ma zgłaszać wyjątek NowyWyjatek to musi podać w deklaracji, że może to zrobić tzn.:

```
void naszaMetoda () throws nowyWyjatek
```

- naszaMetoda sprawdza warunki powstania błędu, jeśli wystąpił błąd – tworzy wyjątek:

```
new NowyWyjatek(..)
```

i sygnalizuje go za pomocą instrukcji throw:

```
throw new NowyWyjatek(ew_param_konstr_z_info_o_bledzie)
```

Przykład - klasa wyjątku NiePoprawnyKod i klasa komponentu PoleKodu

```
class NiePoprawnyKod extends Exception  
{ //klasa wyjątku  
public String info="Poprawny kod ma postać: nn-nnn";  
NiePoprawnyKod(String s)  
{  
info="Niepoprawny kod:"+s+"."+info;  
}  
}  
class PoleKodu extends JTextField { //klasa komponentu  
PoleKodu(){ super();}  
PoleKodu(String s){ super(s);}  
PoleKodu(int n){ super(n);}
```

Przykład – własne wyjątki c.d.

```
public String pobierzKod() throws NiePoprawnyKod{  
final int N=6,P=2;  
String kod=getText();  
boolean poprawny=true;  
char []c=kod.toCharArray();  
if (c.length!=N || c[P]!='-') poprawny=false;  
else for(int i=0;i<N;i++){  
if (i==P) continue;  
if (!Character.isDigit(c[i])) poprawny=false;  
}  
if (!poprawny) throw new NiePoprawnyKod(kod);  
return kod;  
}  
}
```

Przykład – wykorzystanie własnego wyjątku

```
import javax.swing.*;
```

```

import java.awt.*;
import java.awt.event.*;

public class Kod extends JFrame implements ActionListener {
    PoleKodu pk=new PoleKodu(10);
    JPanel glowny=new JPanel(); JTextArea t=new JTextArea();
    public Kod() {
        setSize(320,150);
        //setDefaultCloseOperation(EXIT_ON_CLOSE);
        glowny.setLayout(new BoxLayout(glowny,BoxLayout.Y_AXIS));
        JLabel l1=new JLabel("Wprowadź kod:",JLabel.CENTER);
        JButton b=new JButton("Enter"); b.addActionListener(this);
        glowny.add(l1); glowny.add(pk);
        glowny.add(b); glowny.add(t);
        setContentPane(glowny);
        setVisible(true);
    }
}

```

Przykład – wykorzystanie własnego wyjątku

```

public static void main(String[] args) { new Kod(); }
public void actionPerformed(ActionEvent e){
    String tekst=null;
    try { tekst=pk.pobierzKod(); }
    catch (NiePoprawnyKod ek) { t.setText(ek.info); return; }
    t.setText("Wprowadzono kod:"+tekst);
} }

```

Przykład

```

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

public
class Aplikacja extends JFrame
{
    private JTextArea textArea;
    private JButton button1;
    public Aplikacja()
    {
        ActionListener al = new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                if(e.getSource() == button1)
                    save();
            }
        };
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
    }
}

```

```

        textArea = new JTextArea();
        textArea.setBounds( 0, 0, 260, 200);
        this.add(textArea);
        button1 = new JButton("Zapisz");
        button1.setBounds(100, 240, 100, 20);
        button1.addActionListener(al);
        add(textArea);
        add(button1);
        setSize( 20, 00);
        setVisible(true);
    }
    public void save()
    {
        FileWriter fileWriter = null;
        String text = textArea.getText();
        try
        {
            fileWriter = new FileWriter("test.txt", true);
            fileWriter.write(text, 0, text.length());
            fileWriter.close();
        }
        catch(IOException e)
        {
            //System.out.println("Błąd podczas zapisu pliku.");
        }
    }
    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                new Aplikacja();
            }
        });
    }
}

```

Opis

Tworzymy jeden komponent klasy JTextArea (obiekt textArea) oraz jeden przycisk.

Aplikacja działa w taki sposób, że po kliknięciu przycisku cały tekst z pola tekstowego zostanie zapisany w pliku o nazwie test.txt.

Obsługę zdarzeń związanych z klikaniem przycisku zapewnia nam obiekt klasy anonimowej implementującej interfejs ActionListener, dokładnie tak samo jak w przypadku wcześniej prezentowanych przykładów. Po kliknięciu przycisku jest zatem wywoływana metoda actionPerformed, która sprawdza, czy sprawcą zdarzenia faktycznie jest przycisk button1. Jeśli tak, wywoływana jest metoda save klasy TextArea. Metoda save korzysta z obiektu klasy FileWriter do zapisania zawartości komponentu textArea w pliku (

Pojęcie strumienia

- programowanie operacji we/wy polega na użyciu strumieni, które są obiektami klas strumieniowych z pakietu java.io
 - strumień danych – pojęcie abstrakcyjne, oznacza ciąg danych, do którego dane mogą być dodawane (operacja zapisu do strumienia – strumień wyjściowy) i z którego mogą być pobierane (operacja odczytu ze strumienia – strumień wejściowy)
 - pobranie/zapis danych dotyczy określonych porcji danych
- w Javie są to strumienie bajtowe (atomem operacji jest bajt) i strumienie znakowe (atomem jest znak Unicode – 2 bajty) – cztery hierarchie klas strumieniowych

Przodkowie hierarchii klas strumieniowych

- są to klasy abstrakcyjne (nie można tworzyć bezpośrednio obiektów tych klas) – dostarczają metod m.in. do czytania (read), zapisywania (write), zamykania strumieni (close)

Writer Reader Strumienie znakowe

OutputStream InputStream Strumienie bajtowe Wyjście Wejście

InputStream, OutputStream

- Podklasy InputStream:
 - AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream
- Podklasy OutputStream:
 - ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, PipedOutputStream

Reader, Writer

- Podklasy Reader:
 - BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader
- Podklasy Writer:
 - BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

Zastosowanie strumieni danych

- utworzenie obiektu reprezentującego strumień (np. obiektu klasy FileInputStream dla strumienia we lub obiektu klasy BufferedWriter dla strumienia wy)
- odczyt/zapis danych do/ze strumienia (np. metoda read() dla obiektu FileInputStream lub metoda write() dla obiektu BufferedWriter)
- zamknięcie strumienia metodą close()

STRUMIENIE BAJTOWE

- strumienie bajtowe - podklasy klas abstrakcyjnych InputStream/OutputStream
- FileInputStream/FileOutputStream - klasy podrzędne, reprezentują strumienie bajtowe powiązane z plikami

- `DataInputStream/DataOutputStream` – klasy reprezentujące filtrowane strumienie bajtowe (umożliwiające odczyt danych typu `int`, `float` itp.)
- strumienie plikowe - najczęstsze strumienie bajtowe

Wejściowe strumienie plikowe

- wejściowy strumień plikowy - obiekt `FileInputStream(string)` z argumentem będącym nazwą pliku (może być zgłoszony wyjątek `IOException`)
- `read()` - czyta 1 bajt z pliku (zwraca -1 gdy osiągnięto koniec pliku)
- `read(byte[], int1, int2)` – odczytanie `int2` bajtów ze strumienia do tablicy i umieszczenie ich począwszy od elementu tablicy o indeksie `int1`
- np.:

```
FileInputStream f=new FileInputStream("dane.txt");
int bajt=0;
while(bajt!=-1) {
    bajt=f.read();
    System.out.print(bajt+"");
}
```

Przykład - odczyt bajtów z pliku

```
import java.io.*;
public class CzytajBajty {
    public static void main(String[] args) {
        try { FileInputStream pl=new FileInputStream("D:/beata/java/dane.dat");
            boolean eof=false; int ile=0;
            while (!eof) {
                int input=pl.read();
                System.out.print(input+" ");
                if (input==-1) eof=true; else ile++;
            }
            pl.close();
            System.out.print("\n Wczytano "+ile+" bajtow.");
        } catch(IOException e) { System.out.println("Error -"+e.toString()); }
    }
}
```

Wyjściowe strumienie plikowe

- wyjściowy strumień plikowy – obiekt `FileOutputStream(string)` z argumentem będącym nazwą pliku (może być zgłoszony wyjątek `IOException`); jeśli plik istnieje to jego zawartość zostanie usunięta
- `FileOutputStream(string, true)` – utworzenie strumienia wy, który dopisuje dane do pliku
- `write()` – zapisuje 1 bajt do pliku
- `write(byte[] , int i, int n)` – `byte[]` tablica bajtów do zapisu, `i` – indeks pierwszego elementu do zapisu, `n` – liczba bajtów do zapisu

Przykład - zapis bajtów do pliku

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

```

public class ZapiszBajty {
public static void main(String[] args) {
int []dane={ 71,73,70,56,57,97,15,0,15,0,128,0,0,
255,255,255,0,0,0,44,0,0,0,
0,15,0,15,0,0,2,33,132,127,161,200,
185,205,84,128,241,81,35,175,155,26,
228,254,105,33,102,121,165,201,145,169,
154,142,172,116,162,240,90,197,5,0,59};
try {
FileOutputStream pl=new FileOutputStream("obraz.gif");
for (int i=0;i<dane.length;i++) pl.write(dane[i]);
pl.close();
}
catch(IOException e) { System.out.println("Error -"+e.toString()); }
}
}

```

Filtrowanie strumienia danych

- Filtr - specyficzny typ strumienia, który modyfikuje sposób, w jaki obsługiwany jest bieżący strumień danych

Wykorzystanie filtra:

- utworzenie strumienia powiązanego ze źródłem danych lub miejscem ich zapisu
- utworzenie powiązania filtra ze strumieniem
- zapis/odczyt danych bezpośrednio z filtra a nie ze strumienia
- filtry mogą być zagnieżdżane

Filtry bajtowe – strumienie

buforowane (strumień we)

- Bufor – obszar przeznaczony do przechowywania danych
- Buforowany strumień wejściowy – wypełnia bufor danymi, których program jeszcze nie wykorzystał(program najpierw sprawdza bufor, zanim sięgnie do źródła danych)
- `BufferedInputStream(InputStream)` – tworzy buforowany strumień wejściowy dla strumienia reprezentowanego przez `InputStream`
- `BufferedInputStream(InputStream,int)` – j.w., `int` określa rozmiar bufora
- `read()` – odczytanie 1 bajtu ze strumienia buforowanego
- `read(byte[], int1, int2)` – odczytanie `int2` bajtów ze strumienia do tablicy i umieszczenie ich począwszy od elementu tablicy o indeksie `int1`

Filtry bajtowe – strumienie buforowane (strumień wy)

- `BufferedOutputStream(OutputStream)` – tworzy buforowany strumień wyjściowy dla strumienia reprezentowanego przez `OutputStream`
- `BufferedOutputStream(OutputStream,int)` – j.w., `int` określa rozmiar bufora
- `write(int)` – zapisanie 1 bajtu do strumienia buforowanego (argument `int` z przedziału 0-255, jeśli większy to brana jest reszta z dzielenia całkowitego)
- `write(byte[], int1, int2)` – zapisanie `int2` bajtów z tablicy `byte[]`, począwszy od elementu tablicy o indeksie `int1` do strumienia buforowanego
- dane nie zostaną przekazane do miejsca docelowego dopóki bufor nie zostanie całkowicie wypełniony lub dopóki nie będzie wywołana metoda `flush()`

Przykład – zapis i odczyt, strumienie buforowane

```
import java.io.*;
class ArgStream{ //klasa ArgStream
int start=0;
int stop=25;
ArgStream(int p,int k){ start=p; stop=k; }
boolean writeStream(){
try { FileOutputStream pl=new FileOutputStream("liczby.dat");
BufferedOutputStream buff=new BufferedOutputStream(pl);
for (int out=start;out<=stop;out++){
buff.write(out); System.out.print(" "+out); }
buff.close();
return(true);
} catch(IOException e) {
System.out.println("Wyjatek:"+e.getMessage());
return false; }
}
```

Przykład – zapis i odczyt, strumienie buforowane

```
boolean readStream(){
try { FileInputStream pl=new FileInputStream("liczby.dat");
BufferedInputStream buff=new BufferedInputStream(pl);
int in=0;
do{ in=buff.read();
if(in!=-1) System.out.print(" "+in);
} while (in!=-1);
buff.close();
return(true);
}
catch(IOException e) {
System.out.println("Wyjatek:"+e.getMessage());
return false;
}
} //koniec klasy ArgStream
```

Przykład – zapis i odczyt, strumienie buforowane

```
public class BuforDemo {
public static void main(String[] args) {
int start=0;
int stop=25;
if (args.length>1){
start=Integer.parseInt(args[0]);
stop=Integer.parseInt(args[1]); }
else if (args.length>0)
start=Integer.parseInt(args[0]);
ArgStream as=new ArgStream(start,stop);
System.out.println("\nZapisywanie:");
boolean sukces=as.writeStream();
System.out.println("\nCzytanie:");
```

```
boolean sukces=as.readStream();
} }
```

We/wy strumienie danych

- filtrują istniejące strumienie bajtowe umożliwiając odczyt i zapis bezpośrednio do strumienia danych typu boolean, double, float, int, long oraz short
- `DataInputStream(InputStream)` - strumień we
- `DataOutputStream(OutputStream)` - strumień wy
- metody:
 - `readBoolean()`, `writeBoolean()`
 - `readByte()`, `writeByte()`
 - `readDouble()`, `writeDouble()`
 - `readFloat()`, `writeFloat()`
 - `readInt()`, `writeInt()`
 - `readLong()`, `writeLong()`
 - `readShort()`, `writeShort()`

We/wy strumienie danych

- dodatkowe metody:
 - `readUnsignedByte()`
 - `readUnsignedShort()`
- metody odczytujące ze strumienia we nie zwracają żadnej wartości wskazującej na osiągnięcie końca odczytywania strumienia - można wykorzystać wystąpienie wyjątku `EOFException` (następuje po osiągnięciu końca strumienia)

Przykład - zapis liczb Int

```
import java.io.*;
public class ZapiszInt {
    public static void main(String[] args) {
        try {
            FileOutputStream pl=new FileOutputStream("liczby.dat");
            BufferedOutputStream buff=new BufferedOutputStream(pl);
            DataOutputStream liczby=new DataOutputStream(buff);
            for (int i=1;i<=10;i++) liczby.writeInt(i);
            liczby.close();
        }
        catch(IOException e)
        { System.out.println("Error -"+e.toString());}
    } }
```

Przykład - odczyt liczb Int

```
import java.io.*;
public class OdczytInt {
    public static void main(String[] args) {
        try {
            FileInputStream pl=new FileInputStream("liczby.dat");
            BufferedInputStream buff=new BufferedInputStream(pl);
```

```

DataInputStream liczby=new DataInputStream(buff);
try { while(true) {
int i=liczby.readInt();
System.out.print(" "+i); }
} catch(IOException e) { liczby.close();}
} catch(IOException e) {
System.out.println("Error -"+e.toString());}
} }

```

STRUMIENIE TEKSTOWE

- reprezentowane przez obiekty podklas Reader i Writer
- wszystkie operacje tekstowe powinny być realizowane przez powyższe podklasy a nie strumienie bajtowe

Strumienie tekstowe - odczyt znak po znaku

- klasa FileReader (podklasa InputStreamReader)
 - metody:
 - read() odczyt 1 znaku
 - read(char[], int1, int2) odczyt int2 znaków i umieszczenie w tablicy char począwszy od pozycji int1
 - read zwraca wartość typu int (numeryczny odpowiednik znaku Unicode); przed wyświetleniem na ekran należy rzutować int na char, np.
- ```

FileReader text=new FileReader("znaki.txt");
int znak;
do { znak=text.read();
if (znak!=-1) System.out.print((char) znak); }
while (znak!=-1);
text.close();

```

Strumienie tekstowe – odczyt wierszy

- klasa BufferedReader - odczyt wierszy; należy posiadać obiekt klasy Reader
  - BufferedReader(Reader) - tworzy buforowany strumień znakowy powiązany z obiektem Reader (np.
- FileReader)
  - BufferedReader(Reader, int) - j.w. wyposażony w bufor o rozmiarze int
- metody:
  - read jak dla FileReader
  - readLine() - odczyt całego wiersza (bez znaku końca wiersza); zwraca obiekt klasy String lub null
  - koniec wiersza może być wskazany przez znaki: '\n', '\r',

Przykład - Czytanie wierszy z pliku tekstowego

```

import java.io.*;
import java.io.FileReader;
class Czytajwiersz {

```

```

public static void main(String[] args)
{ try{
FileReader pl=new FileReader("D:\beata\java\ZapiszInt.java");
BufferedReader buff=new BufferedReader(pl);
boolean eof=false;
while (!eof)
{ String linia=buff.readLine();
if (linia==null) eof=true;
else System.out.println(linia);
}
buff.close();
}
catch (IOException e) {System.out.println("Bład zamknięcia pliku."+e); }
}}

```

Strumienie tekstowe - zapis

- klasa FileWriter (podklasa OutputStreamWriter)
- konstruktory:
  - FileWriter(String s)
  - FileWriter(String s,boolean b), gdzie  
b=true (opcjonalnie) - dane ze strumienia dopisywane są na  
końcu pliku,  
b=false - dane zastępują dotychczasową zawartość pliku
- metody:
  - write(int) zapis 1 znaku
  - write(char[], int1, int2) zapis int2 znaków z tablicy char  
począwszy od pozycji int1
  - write(String s, int1, int2) zapis int2 znaków z łańcucha s  
począwszy od pozycji int1

Przykład - zapis do pliku tekstowego

```

import java.io.*;
import java.io.FileWriter;
public class Zapisztxt {
public static void main(String[] args)
{ try{
FileWriter p=new FileWriter("tekst.txt");
for (int i=65;i<91;i++) p.write((char)i);
p.write("\n");
for (int i=97;i<123;i++) p.write((char)i);
p.close();
}
catch (IOException e)
{System.out.println("Bład pliku."+e); }
}
}

```

Strumienie tekstowe – zapis buforowanego strumienia znaków

- klasa BufferedWriter;
- BufferedWriter(Writer) - tworzy buforowany strumień

znakowy powiązany z obiektem Writer (np. FileWriter)

– BufferedWriter(Writer, int) - j.w. z buforem o rozmiarze int

- metody jak dla FileWriter :

– write(int)

– write(char [],int1, int2)

– write(String, int1, int2)

- metoda newLine() - przesyła do strumienia wy domyślny dla danej platformy znak reprezentujący zakończenie wiersza

Inne rodzaje filtrów - klasa File

- klasa File - reprezentuje odwołanie do pliku lub foldera; konstruktory:

– File(String f) - tworzy obiekt klasy File powiązany z folderem f

– File(String1 f, String2 p) - tworzy obiekt klasy

File powiązany z folderem f oraz plikiem p

– File(File f, String p) - tworzy obiekt klasy File powiązany z folderem reprezentowanym przez f oraz plikiem p

Metody klasy File

- boolean exists()

- boolean renameTo(File)

- boolean delete()

- boolean mkdir()

- Powyższe metody mogą generować wyjątek

SecurityException - powinny być wykonywane w konstrukcji try-catch

Rodzaje strumieni

- strumień bajtowe - umożliwiają zapisywanie i odczytywanie danych całkowitych o wartościach 0..255

- strumień tekstowe - umożliwiają zapisywanie i odczytywanie danych tekstowych

- strumień obiektowe (ang. object streams)

– pozwalają na reprezentację danych danych jako integralną część obiektu

– pakiet java.io

– przed zapisem obiektu w postaci pliku na dysku należy dokonać jego konwersji do postaci szeregowej

Serializacja obiektów

- obiekt przesyłany za pomocą strumieni obiektowych musi posiadać zaimplementowany interfejs

Serializable, który wskazuje, że dany obiekt może być przesyłany i przechowywany w postaci szeregowej (serializowanej)

- serializacja oznacza również, że obiekt może egzystować poza programem, który go utworzył (obiekt trwały - ang. persistence)

- z procesu serializacji obiektu można wykluczyć

niektóre jego zmienne (nie są zapisywane pola statyczne oraz pola deklarowane z identyfikatorem transient

44

Wykorzystanie serializacji

- komunikacja między obiektami poprzez gniazdka (ang. sockets)
- zachowanie obiektu (jego stanu i właściwości) do późniejszego wykorzystania przez tę samą lub inną aplikację
- mechanizm serializacji
  - zamiana obiektu na sekwencję bajtów
  - przesłanie sekwencji bajtów
  - odtworzenie oryginalnego obiektu
- do zapisywania/odczytywania obiektów służą klasy (strumienie):
  - ObjectOutputStream
  - ObjectInputStream

Interfejs Serializable

- Możliwość serializacji obiektu - jeśli klasa implementuje interfejs Serializable
- Interfejs Serializable - pusty (brak konieczności przesłaniania metod)
- Obsługa serializacji - w klasach ObjectInputStream i ObjectOutputStream

Zapis/odczyt obiektu do strumienia

Zapis:

- utworzenie obiektu klasy potomnej po OutputStream
- powiązanie obiektu z obiektem klasy ObjectOutputStream
- wywołanie metody writeObject()
- możliwe wystąpienie wyjątku NotSerializableException

Odczyt:

- utworzenie obiektu klasy potomnej po InputStream
- powiązanie obiektu w obiekt klasy ObjectInputStream
- wywołanie metody readObject()

Przykład - aplikacja PiszObiekt

```
import java.io.*;
import java.util.*;
public class PiszObiekt implements Serializable {
 public static void main(String[] args) {
 Calendar cal = Calendar.getInstance();
 Date data = cal.getTime();
 try {
 FileOutputStream p1 = new
 FileOutputStream("czas.obj");
 ObjectOutputStream s = new ObjectOutputStream(p1);
 s.writeObject("Dzisiaj jest: ");
```

```
s.writeObject(data);
s.close();
} catch (IOException e) { }
}
}
```

Przykład - aplikacja CzytajObiekt

```
import java.io.*;
import java.util.*;
public class CzytajObiekt implements Serializable {
 public static void main(String[] args) {
 try {
 FileInputStream p2 = new FileInputStream("czas.obj");
 ObjectInputStream s = new ObjectInputStream(p2);
 String dzisiaj = (String)s.readObject();
 Date data = (Date)s.readObject();
 s.close();
 System.out.println(dzisiaj + data);
 } catch (IOException e) { }
 catch (ClassNotFoundException e) { }
 }
}
```

Jeżeli miesiąc jest mniejszy od 3 to cofnij rok o 1, miesiąc zwiększ o 12

$$d = r + r \setminus 4 - r \setminus 100 + r \setminus 400 + 3m - (2m + 1) \setminus 5 + d + 1$$

- oblicz  $a = (14 - \text{miesiąc}) \setminus 12$ ,

- oblicz  $y = \text{rok} - a$ ,

- oblicz  $m = \text{miesiąc} + 12 * a - 2$ ,

- oblicz  $d = (\text{dzień} + y + y \setminus 4 - y \setminus 100 + y \setminus 400 + (31 * m) \setminus 21) \bmod 7$ ,

jeżeli d jest równe 0 badany dzień wypada w Niedzielę, 1 w Poniedziałek etc.

Sposób działania algorytmu zamieszczono poniżej (uwaga operator "\ " oznacza część całkowitą z dzielenia dwóch liczb).

```
// Runtime.getRuntime().exec("cmd dir");
System.out.println("\033[2J"); // Czyści terminal
System.out.println("\033[0;0f"); // Ustawia kursor w lewym, górnym rogu
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
public class DrugiProgram {
```

```
 public static void main(String[] args) {
 try {
```

```
 BufferedReader reader = new BufferedReader(new InputStreamReader(
 System.in));
```

```

 System.out.println("Jak masz na imie?");
 String imie = reader.readLine();
 System.out.println("Witaj " + imie.toUpperCase() + "!");
} catch (IOException e) {
 System.out.println("Wystapil blad przy czytaniu z klawiatury. " + e);
}

}
}

```

---

Etykiety - do wyświetlania tekstu, który nie ulega zmianie :

Przyciski - do wykonywania działań :

Pola wyboru - pozwalają włączać lub wyłączać różne opcje :

Przyciski opcji - pozwalają wybrać jedną z wielu opcji :

Pola list rozwijalnych - pozwalają wybrać element z dużej grupy :

Pola tekstowe - pozwalają wpisać linijkę tekstu :

Pola list - pozwalają na wybór jednej z wielu opcji :

Pola przewijania - do ustalania pewnej wartości z określonego przedziału

Pola edycji - pozwalają wpisać wiele linii tekstu :

Tworzenie elementów interfejsu

Label etykieta;

etykieta = new Label("To jest tekst etykiety");

add(etykieta);

Menedżery położenia

FlowLayout - domyślny, elementy umieszczane są od lewej do prawej

Grid - applet podzielony jest na wiersze i kolumny, każdy element ma swoją komórkę

Border - applet podzielony jest na strefy N,E,W,S,Center

Card - elementy są umieszczane na kartach



GridBagLayout - dzielony na komórki, do elementów przypisane są "wagi"

Wybór menedżera

metoda : setLayout(.....)

Opis menedżerów

FlowLayout

public void init()

```
{
setLayout(new FlowLayout(FlowLayout.CENTER));
}
```

FlowLayout(wyrównanie,odstęp,odstęp);

# wyrównanie FlowLayout.LEFT

# FlowLayout.CENTER

# FlowLayout.RIGHT

Grid

public void init()

```
{
setLayout(new GridLayout(4,4); }
```

GridLayout(wiersze,kolumny,odst\_pion,odst\_poz);

Border

public void init()

```
{
setLayout(new BorderLayout());
}
```

BorderLayout(odst\_pion,odst\_poz);

add("North",element);

Card

setLayout(new CardLayout());

GridBagLayout

z każdym elementem jest skojarzony obiekt typu GridBagConstraints określający sposób wyświetlania elementu

# pola klasy gridx,gridy - określają położenie komórki w lewym górnym rogu

# gridwidth,gridheight - liczby komórek

# weightx,weighty - wagi

# ....

Dodawanie elementów standardowych do appletu

## Dodawanie etykiet

```
Label etykieta;
etykieta = new Label("Tekst");
add(etykieta);

add(new Label("tekst",wyrównanie);

wyrównanie Label.LEFT
Label.CENTER
Label.RIGHT
```

## Dodawanie przycisków

```
Button przycisk;
przycisk = new Button("OK");
add(przycisk);
```

## Dodawanie pól wyboru

```
Checkbox pole;
pole = new Checkbox("tekst",null,true);
add(pole);
```

## Dodawanie przycisków opcji

```
tworzemy CheckboxGroup();

CheckbocGroup po;
po = new CheckboxGroup();

dodawanie pól
add(new Checkbox("przycisk1",po,true);
```

## Dodawanie list rozwijalnych

```
Choice lista;
lista = new Choice();

dodawanie do listy
lista.addItem("pierwszy");
```

## Dodawanie pól tekstowych

```
TextField pole;
pole = new TextField("tekst",30);
add(pole);

TextField(tekst,max_liczba_znaków);
```

```
pole.setEchoCharakter('*');
```

Obsługa standardowych elementów

```
public boolean action(Event z, Object arg)
{
 // przetwarzanie zdarzeń }

```

Event - przekazuje typ zdarzenia

Object - przekazuje dodatkowe informacje

z.target - przekazuje informacje o klasie elementu, który wygenerował zdarzenie

```
public boolean action(Event z, Object arg)
{
 if (z.target instanceof Checkbox)
 {
 }
 else
 if (z.target instanceof Choice)
 {
 }
 else
 if (z.target instanceof TextField)
 {
 }
 return true;
}

```

Etykiety - nie generują zdarzeń

Przyciski - przekazują nazwę przycisku

Pola wyboru - przekazują wartość logiczną

Przyciski opcji - w Object umieszczają stan przycisków

Pola list - przekazują łańcuch z nazwą wybranej opcji

Pola tekstowe - powodują wygenerowanie zdarzenia w chwili wciśnięcia Enter, w Object przekazują pobrany tekst

```
import java.applet.*;
import java.awt.*;
public class Interfejs extends java.applet.Applet
{
 // pola danych
 Label etykieta;
 Button przycisk;
 Checkbox pole_w;
 Checkbox po1, po2, po3;
 CheckboxGroup grupa;
 Choice lista;
 TextField pole_t;

 // inicjacja
 public void init()

```

```

{
// ustawienie menedżera położenia
setLayout(new FlowLayout(FlowLayout.LEFT));

// utworzenie nowej etykiety
// i dodanie jej do apletu
etykieta = new Label("To jest statyczna etykieta");
add(etykieta);

// utworzenie nowego przycisku
// i dodanie go do apletu
przycisk = new Button("OK");
add(przycisk);

// utworzenie nowego pola wyboru
// i dodanie go do apletu
pole_w = new Checkbox("Pola wyboru przeznaczone są dla wartości logicznych");
add(pole_w);

// utworzenie grupy przycisków opcji
// i dodanie jej do apletu
grupa = new CheckboxGroup();
po1 = new Checkbox("Przycisk1", grupa, true);
po2 = new Checkbox("Przycisk2", grupa, false);
po3 = new Checkbox("Przycisk3", grupa, false);
add(po1);
add(po2);
add(po3);

// utworzenie nowej listy
// i dodanie jej do apletu
Choice lista = new Choice();
lista.addItem("jeden");
lista.addItem("dwa");
lista.addItem("trzy");
lista.addItem("cztery");
add(lista);

// utworzenie nowego pola tekstowego
// i dodanie go do apletu
pole_t = new TextField(20);
add(pole_t);
}

public boolean action(Event zdarzenie, Object arg)
{
if (zdarzenie.target instanceof Button)
{
System.out.print("Naciśnięty przycisk. Argument=");
System.out.println(arg);
}
}

```

```

}
else
if (zdarzenie.target instanceof Checkbox)
{
System.out.print("Zmienione pole wyboru. Nazwa=");
// pobranie i wyświetlenie nazwy pola
String x = ((Checkbox)zdarzenie.target).getLabel();
System.out.print(x);
// wyświetlenie argumentu (true lub false)
System.out.print(" Argument=");
System.out.println(arg);
}
else
if (zdarzenie.target instanceof Choice)
{
System.out.print("Zmieniony stan listy. Argument=");
System.out.println(arg);
}
else
if (zdarzenie.target instanceof TextField)
{
System.out.print("Pole tekstowe. Argument=");
System.out.println(arg);
}
return true;
}
}

```

Dodawanie elementów rozszerzonych

Listy

```

public void init()
{
Lista lista;
lista = new List(5,false);
add(lista);
}

```

dodawanie pozycji na liście

```

lista.addItem("pierwszy");

```

Paski przewijania

```

Scrollbar pasek;
pasek = new Scrollbar(Scrollbar.HORIZONTAL,1,50,1,100);
add(pasek);

```

Pola edycji

```
TextArea pole;
pole = new TextArea("tekst",5,5);
add(pole);
```

Obsługa elementów rozszerzonych

```
public boolean handleEvent(Event z)
{
 if (z.target instanceof List)
 {}
 else
 if (z.target instanceof Scrollbar)
 {}
 else
 if (z.target instanceof TextArea)
 {}
 return true;
}
```

```
public boolean handleEvent(Event z)
{
 if (z.target instanceof List)
 {
 String s = ((List)z.target).getSelectedItem();
 System.out.println(s);
 } return true;
}
```

```
import java.applet.*;
import java.awt.*;
```

```
public class Kontrolki extends java.applet.Applet
{
 // pola danych
 List lista;
 TextArea pole_ed;
 Scrollbar pasek_poz, pasek_pion;

 // inicjacja
 public void init()
 {
 // ustawienie menedżera położenia
 setLayout(new FlowLayout(FlowLayout.LEFT));

 // utworzenie listy i dodanie jej do apletu
 lista = new List();
 lista.addItem("styczeń");
```

```
lista.addItem("luty");
lista.addItem("marzec");
lista.addItem("kwiecień");
lista.addItem("maj");
lista.addItem("czerwiec");
lista.addItem("lipiec");
lista.addItem("sierpień");
lista.addItem("wrzesień");
lista.addItem("październik");
lista.addItem("listopad");
lista.addItem("grudzień");
add(lista);
```

```
// utworzenie pasków przewijania (pionowy i poziomy)
// i dodanie ich do apletu
pasek_poz = new Scrollbar(Scrollbar.HORIZONTAL, 1, 0, 1, 100);
add(pasek_poz);
pasek_pion = new Scrollbar(Scrollbar.VERTICAL, 1, 0, 1, 100);
add(pasek_pion);
```

```
// utworzenie nowego pola edycji
pole_ed = new TextArea("Domyślny tekst", 10, 60);
add(pole_ed);
}
```

```
public boolean handleEvent(Event zdarzenie)
{
 if (zdarzenie.target instanceof List)
 {
 System.out.print("Zdarzenie związane z listą. Wybrano pozycję: ");
 String s = ((List)zdarzenie.target).getSelectedItem();
 System.out.println(s);
 }
 else
 if (zdarzenie.target instanceof Scrollbar)
 {
 System.out.print("Zdarzenie związane z paskiem przewijania. Wartość: ");
 int i = ((Scrollbar)zdarzenie.target).getValue();
 System.out.println(i);
 }
 else
 if (zdarzenie.target instanceof TextArea)
 {
 System.out.print("Zdarzenie związane z polem edycji. Tekst: ");
 String s = ((TextArea)zdarzenie.target).getText();
 System.out.println(s);
 }
 return true;
}
```

}

## Java przeciążanie metod

Przeciążanie metody umożliwia powtórne użycie tej samej nazwy metody. Dzięki temu nie musimy tworzyć za każdym razem innej nazwy dla metody, widać to doskonale na przykładzie Klasy String, która posiada metodę statyczną `valueOf( <<typ>> )`, napisaną w kilku wersjach dla różnych typów danych, jako argumenty:

```
String.valueOf(boolean b);
String.valueOf(char c);
String.valueOf(char[] data);
String.valueOf(double d);
String.valueOf(float f);
String.valueOf(int i);
String.valueOf(long l);
String.valueOf(Object o);
String.valueOf(char[] data, int offset, int count);
```

Metoda `valueOf( <<typ>> )`, została przeciążona w klasie String, jej nazwa pozostała nie zmieniona, lecz parametry w każdej z nich są inne, taki jest bowiem podstawowy warunek na to, aby móc przeciążyć metodę, musi ona różnić się listą parametrów (ich ilością, lub typem).

Zasady przeciążania metod:

- \* Lista argumentów musi być różna od tej z metody jaką chcemy przeciążyć, może różnić się ilością, typem parametrów, lub ilością i typem jednocześnie. Jeżeli natomiast nazwa metody oraz jej parametry będą takie same, jak w oryginale, będziemy mieć do czynienia z przesłonięciem metody (jeżeli sytuacja ma miejsce w sub klasie), lub błędem (jeżeli sytuacja ma miejsce w tej samej klasie w której znajduje się metoda, jaką przesłaniamy),

- \* Typ zwracany przez metodę przeciążoną, oraz jej specyfikator dostępu, mogą być inne niż w oryginale,

- \* Metody przeciążone mogą deklarować nowe wyjątki, lub poszerzać te, które są już zadeklarowane, pod warunkiem, że dziedziczą one po klasie Exception,

- \* Metoda może zostać przeciążona w ramach tej samej klasy, lub sub klasy,

- \* Typ referencji determinuje, która przeciążona metoda zostanie wywołana.

Ostatni punkt domaga się komentarza. W przypadku metod przesłanianych, o tym, która zostanie wykonana, decyduje typ obiektu, na jaki pokazuje referencja. W przypadku metod przeciążanych, odpowiednia metoda wybierana jest na podstawie typu referencji, a nie obiektu.

Wywołania polimorficzne

Podczas wywoływania metod przeciążonych (w przeciwieństwie do metod przesłoniętych), polimorfizm nie odgrywa żadnej roli, wywoływana jest ta metoda, jaką wskaże typ referencji. Co natomiast stanie się, gdy metoda jest zarówno przeciążona, jak i przesłonięta ?



```

class A {
 void write(){
 System.out.println("oryginał");
 }
}

class B extends A {
 void write(){
 System.out.println("przesłonięta");
 }
 void write(String s){
 System.out.println("przeciążona");
 }
}

class Main {
 public static void main(String[] args) {
 A a = new A();
 B b = new B();
 A ab = new B();

 a.write(); // (1) oryginał
 b.write(); // (2) przesłonięta
 ab.write(); // (3) przesłonięta
 b.write("xx"); // (4) przeciążona
 a.write("xx"); // (5) błąd kompilacji
 ab.write("xx"); // (6) błąd kompilacji
 }
}

```

Cała magia dzieje się już podczas wykonywania programu, tzw. wywołania dynamiczne. Chodzi o to że środowisko w jakim uruchomiony jest program sprawdza na bieżąco (przed wykonaniem jakiejkolwiek operacji) jakiego faktycznie typu jest obiekt wskazywany przez referencję. Od tego zależy jaka metoda zostanie wykonana. Powyższy przykład może wydawać się trudny do zrozumienia, ale wystarczy zapamiętać prostą regułę, operator „.” działa polimorficznie, a operator „( )” nie.

1. Referencja pokazuje na obiekt typu A, więc zostanie wykonana metoda write() z klasy A.
2. Referencja pokazuje na obiekt typu B, więc zostanie wykonana metoda write() z klasy B.
3. Referencja pokazuje na obiekt typu B, więc zostanie wykonana metoda write() z klasy B.
4. Referencja pokazuje na obiekt typu B, więc zostanie wykonana metoda write (String s) z klasy B.
5. Błąd kompilacji. Referencja pokazuje na obiekt typu A, ale ta klasa nie posiada metody write przyjmującej parametr typu String.
6. Błąd kompilacji. Dowód na to że operator „( )” nie działa polimorficznie. Referencja pokazuje na obiekt typu B, lecz z uwagi na operator „( )” program usiłuje wywołać metodę write(String s) z klasy A, której ona nie posiada.

## Inne materiały

Przeciążanie metod to definiowanie wielu metod o tej samej nazwie a różniących się listą argumentów. W zasadzie o metodzie przeciążającej – tj. używającej tej samej nazwy, co już istniejąca metoda – można myśleć jak o metodzie zupełnie nowej. Nie ma więc żadnych zasad ograniczających zwracany typ, deklarowane wyjątki czy modyfikatory dostępu, jako to ma miejsce w przypadku redefiniowania (ang. overriding). W zasadzie można by to sformułować następująco: jeśli chcemy zdefiniować metodę która będzie się nazywała tak samo jak inna, już istniejąca metoda to musimy określić inną listę argumentów. To tyle.

Mamy więc w naszej klasie zdefiniowanych kilka metod o tej samej nazwie, skąd więc wiadomo, którą z nich wywołujemy w danej chwili? Naturalnie, na podstawie listy parametrów, tyle, że nie zawsze sprawa jest bardzo prosta. Weźmy dla przykładu następujący program.

```
class Test {
 public static void main(String[] args) {
 Test obj = new Test();

 System.out.println(obj.getString(1));
 }

 String getString(float f) {
 return "float value: " + f;
 }

 String getString(long l) {
 return "long value: " + l;
 }
}
```

Jaki jest efekt jego działania? Która z metod getString() zostanie wywołana? W tym wypadku efektem działania programu będzie napis "long value: 1". Autorzy książki nie podali w rozdziale drugim precyzyjnych reguł, ale intuicyjnie, wywoływana jest ta metoda, której argumenty "bardziej odpowiadają" przekazywanym parametrom. Szczegóły mają się pojawić w dalszych rozdziałach i póki co nie będę wybiegał przed szereg – poczekam. Popatrzmy jednak na jeszcze jeden przykład.

```
class Parent { }

class Child extends Parent { }

class Test {
 public static void main(String[] args) {
 Parent obj = new Child();

 test(obj);
 }
}
```

```

 }

 static void test(Parent p) {
 System.out.println("method for Parent");
 }

 static void test(Child c) {
 System.out.println("method for Child");
 }
}

```

Efekt działania będzie napis "method for Parent", mimo, że obiekt, dla którego wywołano metodę test() jest klasy Child. Powód jest prosty – to, która z przeciążonych metod zostanie wywołana jest określana w czasie kompilacji, kiedy to nie wiadomo jeszcze, jaki będzie rzeczywisty obiekt, a więc o wywoływanej metodzie decyduje typ referencji nie zaś typ obiektu.

## Java przesłanianie metod

Możliwość przesłonięcia metody, mamy zawsze, gdy klasa dziedziczy po klasie bazowej. Wszystkie metody z klasy bazowej oznaczone jako public, protected, oraz default, możemy przesłaniać. Subklasy są uszczegółowieniem klas bazowych, taka sama relacja zachodzi w odniesieniu do przesłanianych metod. Jeżeli metoda z klasy bazowej nie spełnia naszych oczekiwań, możemy ją przesłonić i napisać własną wersję (np. dodanie nowej funkcjonalności). Metody zaimplementowane w klasach bazowych możemy przesłaniać, ale nie jest to konieczne, zupełnie odwrotnie jest w przypadku metod abstrakcyjnych, takie metody musimy zaimplementować w subklasie. Jeżeli subklasa jest również klasą abstrakcyjną, możemy, ale nie musimy implementować metody abstrakcyjnej z klasy nadrzędnej, zasada jest bowiem taka, że implementacja metody abstrakcyjnej, musi zostać zawarta w pierwszej nieabstrakcyjnej subklasie.

```

class Tools {
 String model = "xxx";
 public String getToolModel() {
 return model;
 }
}
class Hammer extends Tools {
 public String getToolModel() {
 return "yyy";
 }
}
public class Main {
 public static void main(String[] args) {
 Tools tool1 = new Tools();
 System.out.println(tool1.getToolModel()); // dostaniemy napis xxx

 Tools tool2 = new Hammer(); // dostaniemy napis yyy, przykład działania
 // polimorfizmu, wykonana zostanie metoda przesłonięta w klasie Hammer
 System.out.println(tool2.getToolModel());
 }
}

```

```

 Hammer hammer = new Hammer();
 System.out.println(hammer.getToolModel()); // dostaniemy napis yyy
 }
}

```

Zakres przesłanianej metody nie może ulec zawężeniu w subklasie, ale może zostać poszerzony. Jeżeli metoda `getToolModel()` z klasy `Tools` jest publiczna, to podczas przesłaniania jej w subklasie `Hammer`, nie można zmienić specyfikatora dostępu, spowoduje to błąd kompilatora.

Zasady przesłaniania metod:

- \* Przesłaniająca metoda musi mieć taką samą nazwę, oraz ilość i typ argumentów, jakich oczekuje, jeżeli któryś z poprzednich warunków nie jest spełniony, metoda zostanie przeciążona (o tym później), a nie przesłonięta,

- \* Typ zwracany musi być taki sam, jak w oryginale, lub ewentualnie jego podtypem,

- \* Specyfikator dostępu metody przesłanianej nie może zostać zawężony (z `public` na `protected`), ale może zostać poszerzony (z `protected` na `public`),

- \* Metody instancji klasy bazowej mogą być przeciążone tylko w klasach dziedziczących po tej klasie,

- \* Metoda oznaczona jako „`final`” lub „`static`”, nie może zostać przesłonięta,

- \* Przesłonięta metoda może rzucać dowolne wyjątki, które dziedziczą po klasie `RuntimeException` (o wyjątkach opowiem później), niezależnie od tego czy i jakiego typu wyjątki zadeklarowane są w metodzie przesłanianej,

- \* Przesłonięta metoda nie może zwracać wyjątków, które dziedziczą po klasie `Exception`,

- \* Jeżeli przesłonięta metoda rzuca wyjątek, który dziedziczy po klasie `Exception`, a metoda, która ją przesłania nie, kompilator i tak uważa, że takiego wyjątku należy się spodziewać (zupełnie tak, jakby metoda przesłaniająca miała taką funkcjonalność zaimplementowaną).

Dostęp do przesłanianej metody

Bywa tak, że to, co zostało już zaimplementowane w metodzie jaką chcemy przesłonić wymaga jedynie drobnych usprawnień, np dodania kilku funkcjonalności do tego, co już zostało napisane. W takich wypadkach nie musimy powielać kodu, wystarczy wywołać metodę z nadklasy i dopisać brakujące linijki. Aby mieć dostęp do metody z klasy bazowej w metodzie, która ją przesłania musimy użyć operatora `super`:

```

class Tools {
 String model = "xxx";
 public String getToolModel() {
 return model;
 }
}
class Hammer extends Tools {
 public String getToolModel() {
 return super.getToolModel() + "yyy";
 }
}

```

```

class Main extends Hammer {
 public static void main(String[] args) {
 Main c = new Main();
 System.out.println(c.getToolModel()); // na ekran zostanie wypisane:
 xxxyyy
 }
}

```

### **Przykład**

```

public class TryCatch
{
 public static void main(String args[])
 {
 Tablica tab = new Tablica();
 int wart = tab.getElement(20);
 System.out.println("Element nr 20 to: " + wart);
 }
}

class Tablica
{
 int tab[];

 public Tablica()
 {
 tab = new int[5];
 }

 int getElement(int index)
 {
 int val = 0;

 try {
 val = tab[index];
 }
 catch(ArrayIndexOutOfBoundsException e) {
 System.out.println("Nie ma elementu o numerze 20!");
 System.exit(-1);
 }
 return val;
 }
}

```

### **Przykład 2**

```

public class Throws

```

```

{
 public static void main(String args[])
 {
 Tablica2 tab = new Tablica2();
 int value = 0;

 try {
 value = tab.getElement(20);
 }
 catch(ArrayIndexOutOfBoundsException e) {
 System.out.println("Nie ma elementu o numerze 20!");
 System.exit(-1);
 }
 System.out.println("Element nr 20 to: " + value);
 }
}

class Tablica2
{
 int tab[];

 public Tablica2()
 {
 tab = new int[5];
 }

 int getElement(int index) throws ArrayIndexOutOfBoundsException
 {
 return tab[index];
 }
}

```

Przechwytyując wyjątki w Javie ważne jest, aby kolejność procedur obsługi wyjątków zaczynała się od tych bardziej szczegółowych. W takim układzie procedurę `catch(Exception e){}` należy umieścić na końcu listy procedur, gdyż w przeciwnym wypadku pozostałe procedury zostaną pominięte.

### **Tablice nieregularne**

Tablice wielowymiarowe takiej samej ilości komórek w danym wierszu. Przy tworzeniu takich struktur czeka nas więcej pracy niż w przypadku tablic regularnych, gdyż prawdopodobnie każdy wiersz trzeba będzie tworzyć oddzielnie.

Deklaracja takiej tablicy wygląda następująco:

```
int[][] tab
```

Ta deklaracja tworzy zmienną tablicową o nazwie `tab`, której można przypisywać tablice dwuwymiarowe przechowujące liczby typu `int`. Teraz należy utworzyć tablice, która będzie

mogła przechowywać tablice jednowymiarowe liczby typu int, wykorzystać należy więc operator new w postaci:

```
new int[4][]
```

Zatem deklaracja i jednoczesna inicjalizacja zmiennej tab wyglądać będzie następująco:

```
int[][] tab = new int[4][];
```

Teraz kolejnym elementom: tab[0], tab[1], tab[2] i tab[3], trzeba przypisać nowo utworzone tablice jednowymiarowe liczb typu int, tak, aby w komórce tab[0] znalazła się tablica czteroelementowa, tab[1] dwuelementowa, tab[2] jednoelementowa, tab[3] trójelementowa. Trzeba więc wykonać ciąg instrukcji:

```
tab[0] = new int[4];
tab[1] = new int[2];
tab[2] = new int[1];
tab[3] = new int[3];
```

To wszystko, cała tablica jest gotowa. Można zacząć wypełniać ją danymi.  
program 2.28

```
class Main
{
 public static void main (String args[])
 {
 int[][] tab = new int[4][];
 tab[0] = new int[4];
 tab[1] = new int[2];
 tab[2] = new int[1];
 tab[3] = new int[3];
 int licznik = 1;
 for(int i = 0; i < tab.length; i++){
 for(int j = 0; j < tab[i].length; j++){
 tab[i][j] = licznik++;
 }
 }
 for(int i = 0; i < tab.length; i++){
 System.out.print("tab[" + i + "] = ");
 for(int j = 0; j < tab[i].length; j++){
 System.out.print(tab[i][j] + " ");
 }
 System.out.println("");
 }
 }
}
```

### Przykład 3

Przykład przedstawia poprawną hierarchię obsługi wyjątków

```

public class Hierarchia
{
 public static void main (String args[])
 {
 Tablica tab = new Tablica();
 int value = 0;

 try {
 value = tab.getElement(20);
 }
 catch(ArrayIndexOutOfBoundsException e) {
 System.out.println("Nie ma elementu o numerze 20!");
 System.exit(-1);
 }
 catch(Exception e) {
 System.out.println("Jakiś błąd!");
 System.exit(-1);
 }
 System.out.println("Element nr 20 to: " + value);
 }
}

```

Składnia bloku obsługi wyjątków:

### Przykład 3

```

class ThreeException extends Exception {}

public class Finally
{
 static int count = 0;

 public static void main(String[] args)
 {
 while(true) {
 try {
 // Przy postinkrementacji zero jest już na początku
 if(count++ == 0) throw new ThreeException();
 System.out.println("Nie ma wyjątku");
 }
 catch(ThreeException e) {
 System.err.println("ThreeException");
 }
 finally {
 System.err.println("Jestem w sekcji finally");
 if(count == 2) break; // wyjdź z "while"
 }
 }
 }
}

```



```
}
}
```

Przykład oraz zadanie na Instrukcja try ... Catach....

| Nr linii<br>algorytmu | n | S | P | K | s←s+p | i | p← p*k   |
|-----------------------|---|---|---|---|-------|---|----------|
|                       | 3 |   |   |   |       |   |          |
| 1                     | 3 | 1 | 1 |   |       |   |          |
| 2                     | 3 | 1 | 1 | 1 |       |   |          |
| 3                     | 3 |   |   |   | 2=1+1 |   |          |
| 4                     | 3 |   |   |   |       | 1 |          |
| 5                     | 3 |   |   |   |       |   | 1=1*1    |
| 2                     | 3 |   |   | 2 |       |   |          |
| 3                     | 3 |   |   |   | 3=2+1 |   |          |
| 4                     | 3 |   |   |   |       | 1 |          |
| 5                     | 3 |   |   |   |       |   | 2=1*2    |
| 4                     | 3 |   |   |   |       | 2 |          |
| 5                     | 3 |   |   |   |       |   | 4=2*2    |
| 2                     | 3 |   |   | 3 |       |   |          |
| 3                     | 3 |   |   |   | 7=3+4 |   |          |
| 4                     | 3 |   |   |   |       | 1 |          |
| 5                     | 3 |   |   |   |       |   | 12=4*3   |
| 4                     | 3 |   |   |   |       | 2 |          |
| 5                     | 3 |   |   |   |       |   | 36=12*3  |
| 4                     | 3 |   |   |   |       | 3 |          |
| 5                     | 3 |   |   |   |       |   | 108=36*3 |