

UNIwersytet Ekonomiczny w Krakowie

Praktyka programowania imperatywno – obiektowego

Janusz Stal
Janusz Sztorc

Kraków 2009

Spis treści

WSTĘP	5
ROZDZIAŁ 1 ŚRODOWISKO PROGRAMISTYCZNE.....	7
Wprowadzenie do zagadnień	7
Podstawowe pojęcia.....	7
Zestaw narzędzi	8
Tworzenie programu	8
Kompilacja programu	8
Uruchomienie programu	8
Pierwszy program	8
Praca z wierszem poleceń	9
Wykorzystanie IDE.....	10
Stosowane konwencje.....	10
Wejście i wyjście	10
Pytania sprawdzające	10
Zadania do wykonania	11
ROZDZIAŁ 2 TYPY DANYCH, ZMIENNE I OPERATORY	13
Wprowadzenie do zagadnień	13
Typy danych	13
Literały.....	13
Zmienne	14
Stałe	14
Operatory.....	14
Konwersja i rzutowanie typów	15
Łańcuchy znaków	15
Tablice	15
Typ wyliczeniowy	16
Pytania sprawdzające	16
Zadania do wykonania	17
Podstawowe operacje na zmiennych	17
Parametry wiersza poleceń	20
Odczyt danych z konsoli	21
ROZDZIAŁ 3 STEROWANIE PRZEBIEGIEM WYKONANIA PROGRAMU	23
Wprowadzenie do zagadnień	23
Blok instrukcji	23
Instrukcja warunkowa.....	24
Instrukcja wielokrotnego wyboru	24
Pętle nieokreślone	25
Pętle określone.....	26
Przerywanie działania instrukcji sterowania	26
Pytania sprawdzające	27
Zadania do wykonania	28
Przetwarzanie warunkowe	28
Przetwarzanie iteracyjne	30

ROZDZIAŁ 4 TWORZENIE OBIEKTÓW I METOD	37
Wprowadzenie do zagadnień	37
Zmienne typu prostego	37
Zmienne typu klasowego	37
Tworzenie obiektów	38
Usuwanie obiektów	39
Tworzenie metod	39
Zwracanie wartości przez metody	39
Wywołanie metod	40
Przekazywanie danych do metod	40
Modyfikatory dostępu do metod	40
Przeciążanie metod	41
Użycie klas bibliotecznych	41
Klasy opakowujące	42
Pytania sprawdzające	42
Zadania do wykonania	43
Tworzenie metod	43
Wykorzystanie klas bibliotecznych	47
 ROZDZIAŁ 5 KLASY I OBIEKTY	 50
Wprowadzenie do zagadnień	50
Klasy	50
Obiekty	50
Tworzenie klasy	50
Pola klasy	51
Metody klasy	51
Konstruktor	52
Tworzenie obiektów	52
Pojęcia dodatkowe	53
Metody dostępne i modyfikujące	54
Słowo kluczowe this	54
Określanie widoczności składowych klasy	54
Pytania sprawdzające	55
Zadania do wykonania	56
 ROZDZIAŁ 6 KOMPOZYCJA I DZIEDZICZENIE	 63
Wprowadzenie do zagadnień	63
Kompozycja	63
Dziedziczenie	64
Polimorfizm	67
Pakiety	67
Pojęcia dodatkowe	68
Klasa Object	68
Operator instanceof	68
Słowo kluczowe super	68
Pytania sprawdzające	68
Zadania do wykonania	69
Kompozycja	69
Dziedziczenie i polimorfizm	70
Metody klasy Object	75
Tworzenie pakietów	75
 ROZDZIAŁ 7 INTERFEJSY I KLASY ABSTRAKCYJNE	 77
Wprowadzenie do zagadnień	77
Interfejsy	77
Klasy i metody abstrakcyjne	79
Klasy i metody finalne	80

Klasy wewnętrzne	80
Pytania sprawdzające	80
Zadania do wykonania	82
Interfejsy i klasy abstrakcyjne	82
Klasy wewnętrzne i finalne	86
ROZDZIAŁ 8 STRUMIENIE, PLIKI I OBSŁUGA BŁĘDÓW	89
Wprowadzenie do zagadnień	89
Wyjątki i ich struktura	89
Wyjątki kontrolowane i niekontrolowane	90
Instrukcja try..catch	91
Wyrzucanie wyjątku	92
Tworzenie nowej klasy wyjątku	92
Strumień danych	92
Strumień plikowe	93
Strumień sieciowe	94
Pliki i katalogi	94
Pytania sprawdzające	94
Zadania do wykonania	95
Obsługa błędów	95
Pliki i katalogi	97
Strumień plikowe	99
Strumień sieciowe	101
ROZDZIAŁ 9 APLETY	102
Wprowadzenie do zagadnień	102
Pytania sprawdzające	104
Zadania do wykonania	105
Pozycjonowanie tekstu	105
Figury geometryczne	106
Wykresy	107
ROZDZIAŁ 10 ELEMENTY GRAFICZNEGO INTERFEJSU UŻYTKOWNIKA	110
Wprowadzenie do zagadnień	110
Główne okno aplikacji	110
Kontenery	111
Komponenty	113
Menadżery rozkładu	113
Obsługa zdarzeń	115
Elementy graficzne	117
Pytania sprawdzające	118
Zadania do wykonania	119
BIBLIOGRAFIA	126
SPIS RYSUNKÓW	127
SPIS TABEL	128

Wstęp

W dobie społeczeństwa informacyjnego umiejętność tworzenia oprogramowania stanowi istotny element wiedzy pożądaną na rynku pracy. Niestety, przeprowadzane okresowo badania wskazują, iż znaczna liczba pracodawców narzeka na niewielkie umiejętności praktyczne, jakimi dysponują absolwenci opuszczający mury szkół, czy uczelni. Celem niniejszej publikacji jest dostarczenie zarówno studentom jak i nauczycielom materiałów umożliwiających rozwijanie umiejętności praktycznego tworzenia programów komputerowych. Z tego powodu główny nacisk został położony na realizację zadań, ograniczając rozważania teoretyczne do niezbędnego minimum.

Zawarty w pracy materiał oparty został na języku programowania Java, powszechnie wykorzystywanym nie tylko w celach edukacyjnych, ale przede wszystkim w zastosowaniach biznesowych. Dodatkowym atutem przemawiającym za wyborem tego języka jest możliwość prowadzenia zajęć praktycznie na dowolnej platformie programowej, nie ograniczając się wyłącznie do systemu operacyjnego Windows. Nie bez znaczenia pozostaje również szeroka dostępność narzędzi, w większości bezpłatnych, umożliwiających efektywne tworzenie aplikacji w tym języku, a także bogata literatura przedmiotu dostępna zarówno w wersji tradycyjnej, jak i elektronicznej.

Książkę tą kierujemy do uczniów i studentów rozpoczynających edukację związaną z tworzeniem oprogramowania. Tematyka zawarta w niniejszej książce porusza podstawowe kwestie dotyczące programowania imperatywnego oraz obiektowego. Zrealizowany materiał, w naszym zamierzeniu, powinien stanowić podstawę dla dalszego rozwijania praktycznych umiejętności programowania.

Praca została podzielona na 10 rozdziałów. Każdy z nich powinien zostać zrealizowany podczas jednej jednostki zajęć laboratoryjnych (2 x 45 minut). Pozwala to na zrealizowanie całości materiału w przeciągu jednego semestru studiów. Zwykle zajęcia laboratoryjne prowadzone są równolegle wykładami, stanowiącymi istotne wsparcie dla prowadzonych ćwiczeń praktycznych.

Rozdział pierwszy pełni w pracy specyficzną rolę. Został on poświęcony na omówienie zagadnień związanych ze środowiskiem, w jakim rozwijane są aplikacje tworzone w języku programowania Java. Zawarte w nim informacje będą pomocne w szczególności tym uczniom, czy studentom, którzy dysponują własnym sprzętem komputerowym, umożliwiając im prawidłowe przygotowanie i skonfigurowanie stanowiska pracy. Została przyjęta zasada, iż na tym początkowym etapie programowania wykorzystywane będą jedynie podstawowe narzędzia, bez stosowania dostępnych zintegrowanych środowisk do tworzenia i rozwoju aplikacji.

Materiał zawarty w rozdziale drugim i trzecim koncentruje się na zapoznaniu studenta z tajnikami programowania imperatywnego. Poruszone zostały przede wszystkim zagadnienia dotyczące deklaracji zmiennych, stosowania instrukcji przypisania, a także omówione zostały konstrukcje językowe sterujące kolejnością wykonywania poszczególnych instrukcji programu. Pozostałe rozdziały niniejszego opracowania wprowadzają w tematykę programowania obiektowego.

Każda jednostka tematyczna posiada jednolitą strukturę, w skład której wchodzi określenie celu realizowanego materiału, krótkie teoretyczne wprowadzenie do zagadnień, pytania sprawdzające opanowanie materiału teoretycznego oraz zbiór praktycznych zadań do wykonania. Początek rozdziału

poświęcony jest zawsze na omówienie zagadnień, których praktyczna realizacja występuje na końcu jednostki. Nie należy zapominać, iż główny nacisk położony został na część praktyczną, stąd część teoretyczną należy traktować jedynie jako wprowadzenie do omawianych zagadnień, czy też ich usystematyzowanie. Pożądane jest, aby student zapoznał się z tematyką realizowaną w rozdziale uczestnicząc w prowadzonych równolegle wykładach lub też korzystając z bogatej literatury, której wykaz został dołączony na końcu niniejszej pracy. Ze względu na zróżnicowany poziom wiedzy studentów część zadań została przedstawiona wraz częściowym lub całkowitym rozwiązaniem.

Mamy nadzieję, że przedstawione tu propozycje rozwiązań będą pomocne zarówno studentom, jak i nauczycielom prowadzącym zajęcia związane z nauczaniem podstaw programowania komputerów. Jednocześnie będziemy wdzięczni za wszelkie uwagi, a także propozycje, które pozwolą dostosować opracowane materiały do oczekiwań zarówno uczniów, studentów, ale także ich przyszłych pracodawców.

Autorzy

Rozdział 1

Środowisko programistyczne

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- prawidłowo tworzyć kod źródłowy programów w języku Java,
- uruchomić prostą aplikację z wykorzystaniem konsoli lub środowiska programistycznego,
- korzystać z dokumentacji.

Wprowadzenie do zagadnień

Java to szybko rozwijający się język programowania wysokiego poziomu stworzony przez firmę Sun Microsystems™, jak również platforma służąca do uruchamiania aplikacji. Jego podstawowe cechy to:

- pełna obiektowość,
- niezależność od architektury (systemu operacyjnego, procesora),
- funkcjonalność – automatyczne zarządzanie pamięcią, obsługa wyjątków, wielowątkowość, obsługa sieci, tworzenie aplikacji z wykorzystaniem GUI (ang. *Graphical User Interface*), tworzenie apletów (aplikacji uruchamianych w środowisku przeglądarki internetowej),
- niezawodność i bezpieczeństwo tworzonych aplikacji.

Szersze omówienie zarówno właściwości języka, jak i środowiska w którym aplikacje są tworzone i uruchamiane, dostępne jest pod adresem <http://java.sun.com/docs/white/langenv/>

PODSTAWOWE POJĘCIA

W praktyce spotkać się można z wieloma pojęciami i nazwami związanymi z tworzeniem i uruchamianiem aplikacji w języku programowania Java. Poniżej zostały przedstawione najczęściej występujące:¹

JDK (ang. *Java Development Kit*)² – zestaw narzędzi pozwalających na utworzenie programu, w skład których wchodzi kompilator, interpreter kodu bajtowego, przeglądarka umożliwiająca uruchamianie apletów, czy dodatkowe programy narzędziowe,

Kompilator (ang. *compiler*) – program dokonujący zamiany kodu źródłowego na kod zrozumiały dla wirtualnej maszyny Javy, tzw. kod bajtowy (ang. *bytecode*),

JVM (ang. *Java Virtual Machine*) – Wirtualna Maszyna Javy stanowiąca "wirtualny komputer", na którym uruchamiane są programy napisane w języku Java,

JRE (ang. *Java Runtime Environment*) – Wirtualna Maszyna Javy wraz ze zbiorem standardowych klas; JRE jest niezbędne do uruchomienia jakiegokolwiek programu utworzonego w języku Java,

API (ang. *Application Programming Interface*) – kolekcja gotowych do użycia komponentów; w przypadku języka Java jest to zbiór klas i interfejsów pogrupowanych w odpowiednie pakiety,

IDE (ang. *Integrated Development Environment*) – zintegrowane środowisko programistyczne służące do tworzenia, modyfikowania oraz testowania programu,

¹ Zobacz również: <http://java.sun.com/new2java/programming/learn/unravelingjava.html>

² W zależności od środowiska, w którym uruchamiane będą aplikacje utworzone w Javie, zdefiniowanych zostało szereg specyfikacji języka: SE (ang. *Standard Edition*), ME (ang. *Micro Edition*), EE (ang. *Enterprise Edition*).

Konsola, wiersz poleceń – elementy umożliwiające komunikację użytkownika z komputerem za pomocą poleceń tekstowych.

ZESTAW NARZĘDZI

Przygotowanie aplikacji wymaga użycia narzędzi umożliwiających lub wspomagających proces tworzenia kodu źródłowego, jego kompilacji oraz uruchomienia. Do niezbędnych zaliczyć można:

- Java SE Development Kit (<http://java.sun.com/javase/downloads/>),
- znakowy edytor tekstu (np. notepad, notepad++, vi, jedit) lub zintegrowane środowisko programistyczne (np. JCreator, NetBeans, Eclipse),
- dokumentacja, zawierająca szczegółowy opis wszystkich klas dostępnych wraz z Javą (ang. *Java API*) (<http://java.sun.com/reference/api/>).

TWORZENIE PROGRAMU

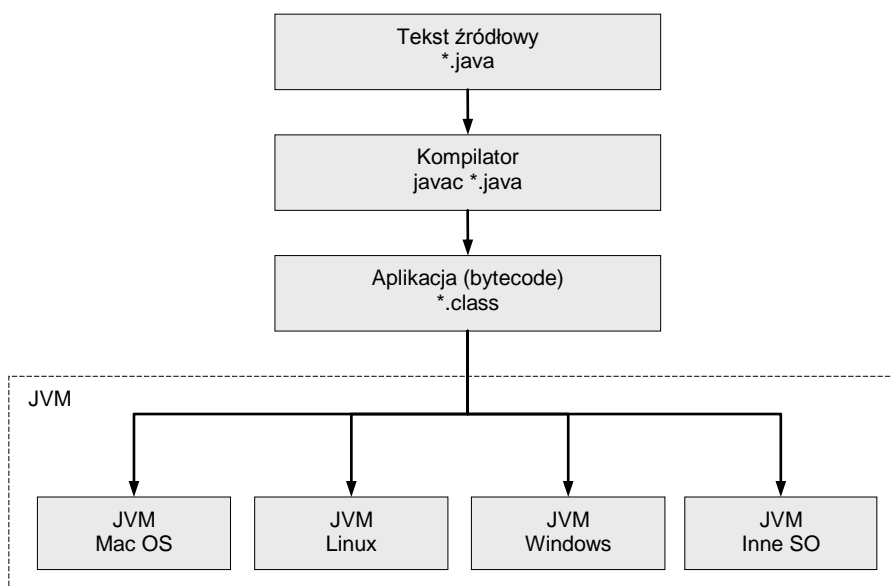
Edycja kodu źródłowego programu zrealizowana może zostać w dowolnym edytorze znakowym. Utworzony w ten sposób plik źródłowy w języku Java posiada rozszerzenie `.java` (np. `Test.java`).

KOMPILACJA PROGRAMU

Zadaniem kompilatora jest zamiana kodu źródłowego programu na kod wykonywany przez komputer. W wyniku kompilacji powstaje program w postaci pliku wykonywalnego z rozszerzeniem `.class` (np. `Test.class`).

URUCHOMIENIE PROGRAMU

Wykonanie programu sprowadza się do wydania komendy `java` wraz z nazwą programu (np. `java Test`). Program zostanie uruchomiony za pomocą Wirtualnej Maszyny Javy.



Rys. 1. Etapy tworzenia i uruchamiania aplikacji.

PIERWSZY PROGRAM

W skład aplikacji tworzonych w języku Java może wchodzić wiele szereg plików źródłowych. W każdym z nich wyróżnić można szereg składowych stanowiących o przyszłym działaniu programu. Typowe elementy każdego pliku to:

- opcjonalny komentarz początkowy, zawierający nazwę autora, opis programu, czy też sposób jego uruchamiania,
- instrukcje deklaracji pakietu bądź/i instrukcje importu użytych klas,
- deklaracje interfejsu lub klasy.


```

/*
 * PierwszyProgram
 * autor: Jan Kowalski (c) 2009
 *
 * Wyświetlenie napisu na konsoli.
 *
 * Uruchomienie: java PierwszyProgram
 *
 */

public class PierwszyProgram {
    public static void main(String[] args) {
        // wyświetlenie prostego napisu
        System.out.println("Pierwszy program w Javie");
    }
}

```

W skład powyższego programu wchodzi klasa o nazwie `PierwszyProgram` zawierająca metodę `main()`, która wyznacza początek działania programu. Jego zadaniem jest wyprowadzenie na konsolę (wyświetlenie na monitorze) przykładowego tekstu, co jest realizowane dzięki użyciu metody `System.out.println()`, argumentem, której jest ciąg znaków do wyświetlenia.

Nagłówek metody w języku Java zawiera informacje o nazwie, modyfikatorze dostępu, typie zwracanej wartości oraz argumentach przekazywanych podczas jej wywołania. Ciąg instrukcji zawartych w metodzie ujęty jest w nawiasy klamrowe, oraz oddzielony średnikami. Poszczególne elementy powyższego programu to:

- `/* ... */` – komentarz blokowy (obejmuje zazwyczaj kilka linii), pomijany podczas kompilacji,
- `public` – modyfikator dostępu³ (określa zakres widoczności klasy),
- `class` – początek definicji klasy,
- `PierwszyProgram` – nazwa klasy (zgodna z nazwą pliku źródłowego),
- `static` – kategoria metody oznaczająca, iż można ją wywołać bez konieczności tworzenia obiektu klasy `PierwszyProgram`,
- `void` – typ wartości zwracanej przez metodę (w tym przypadku metoda nie zwraca żadnej wartości),
- `main()` – nazwa metody, od której rozpoczyna się działanie programu⁴, zdefiniowana w klasie `PierwszyProgram`,
- `(String[] args)` – argumenty metody `main()`; kod metody umieszczony jest zawsze pomiędzy nawiasami klamrowymi,
- `// ...` – komentarz wierszowy⁵, pomijany podczas kompilacji programu,
- `System.out` – użycie klasy `System` wraz ze standardowym strumieniem wyjściowym (skojarzonym domyślnie z ekranem monitora),
- `println()` – metoda umieszczająca łańcuch znaków w strumieniu wyjściowym.

PRACA Z WIERSZEM POLECEŃ

Proces przygotowania kodu programu, jego kompilacji i uruchomienia może zostać przeprowadzony przy użyciu wiersza poleceń lub z wykorzystaniem zintegrowanego środowiska programistycznego (IDE). W pierwszym przypadku należy:

- utworzyć plik z kodem źródłowym w dowolnym edytorze tekstowym,

```
notepad PierwszyProgram.java
```

- skompilować kod źródłowy (powstanie plik z rozszerzeniem `.class`),

```
javac PierwszyProgram.java
```

- uruchomić program,

³ Modyfikatory dostępu zostały omówione szczegółowo w dalszej części pracy.

⁴ Metoda od której rozpoczyna się działanie programu zawiera nagłówek

```
public static void main(String[] args)
```

⁵ Istnieje jeszcze jeden rodzaj komentarza, tak zwany komentarz dokumentacyjny (<http://java.sun.com/j2se/javadoc/>).

```
java PierwszyProgram
```

WYKORZYSTANIE IDE

Zintegrowane środowisko programistyczne umożliwia wykonanie wszelkich czynności wchodzących w skład procesu tworzenia programu, począwszy od utworzenia kodu źródłowego, jego kompilacji oraz uruchomienia programu. W przykładowym IDE JCreator proces ten sprowadza się do:

- utworzenia pliku źródłowego (File → New → File → Java Classes → Empty Java File),
- określenia nazwy i lokalizacji pliku,
- edycji kodu programu w oknie edycyjnym,
- kompilacji kodu źródłowego (Build → Build File),
- uruchomienia programu (Run → Run File).

STOSOWANE KONWENCJE

Tworząc program dobrze jest przestrzegać kilku prostych zasad dotyczących stosowanego nazewnictwa klas, metod, czy zmiennych. Ułatwia to znacznie późniejszą analizę kodu źródłowego. Warto również pamiętać, aby używane nazwy czytelnie identyfikowały elementy kodu programu, a jednocześnie nie były przesadnie zwięzłe⁶. Poniżej przedstawione zostały wybrane zasady, którymi należy się kierować tworząc kod programu.

Nazwy klas, interfejsów:

- rozpoczynają się z wielkiej litery,
- nie zawierają znaku podkreślenia ("_"),
- kolejne wyrazy składające się na nazwę rozpoczynają się z wielkiej litery,
- przykłady: Klient, KlientBanku.

Nazwy zmiennych i metod:

- rozpoczynają się od małej litery,
- nie zawierają znaku podkreślenia,
- kolejne wyrazy składające się na nazwę rozpoczynają się z wielkiej litery,
- nazwa metody powinna określać czynność (zazwyczaj stosuje się parę wyrazów czasownik-rzeczownik, bądź też sam czasownik),
- przykłady: cena, cenaTowaru, dodaj(), dodajPracownika().

Nazwy stałych:

- składają się wyłącznie z wielkich liter,
- kolejne wyrazy oddzielane są znakiem podkreślenia,
- przykłady: PODATEK, PODATEK_VAT

Szczegółowe informacje na temat konwencji stosowanych w kodzie źródłowym programów w języku Java (np. wcięcia, komentarze itp.) dostępne są w sieci Internet⁷.

WEJŚCIE I WYJŚCIE

Standardowe wejście i wyjście związane jest z aplikacją, w której wydawane są komendy (okno terminala). Przekazywanie danych do programu może odbywać się za pomocą dodatkowych parametrów umieszczonych w wierszu poleceń podczas wywołania programu, bądź też korzystając z dostępnych klas pobierających dane od użytkownika w trakcie pracy programu (np. klasa Scanner). Wyświetlanie informacji odbywa się dzięki użyciu klas związanych ze standardowym strumieniem wyjścia. Wykorzystywane są tu przede wszystkim metody print(), println() oraz printf() (np. System.out.print("Java")).

Pytania sprawdzające

1. Wymień główne zalety języka Java.
2. Scharakteryzuj sposób tworzenia i uruchamiania programów tworzonych w języku Java.
3. Wskaż różnice pomiędzy JDK i JRE.

⁶ Należy zwrócić uwagę, iż składnia języka Java rozróżnia wielkość stosowanych znaków (wielkość liter).

⁷ Zobacz: <http://java.sun.com/docs/codeconv/>

4. Określ różnice występujące pomiędzy zwykłym edytorem znakowym, a IDE.
5. Jakie rozszerzenie nazwy posiadają pliki źródłowe, a jakie pliki skompilowane?
6. Czym jest i jaką funkcję pełni Wirtualna Maszyna Javy?
7. Które z wymienionych zmiennych nie są zgodne z przyjętą konwencją tworzenia nazw?

KolorOczu, wzrost, wiekOsoby, umyjTwarz, Gimnastykaporanna, CechaCharakteruKobiety, Licznik

8. Z jakich elementów składa się kod źródłowy?
9. Czy tworząc programy w Javie należy zwracać uwagę na wielkość znaków?
10. Zapoznaj się z dokumentacją do Javy. Sprawdź opis klasy `System` oraz metod służących do wyświetlenia informacji na standardowym wyjściu (np. `print()`, `println()` itd.). Czym różnią się te metody?

Zadania do wykonania

Zadanie 1 – Ustalenie wersji Javy

Sprawdź, czy posiadasz zainstalowany JDK oraz uzyskaj informację o numerze wersji kompilatora i interpretera Javy.

Rozwiązanie

Przejdź do trybu wiersza poleceń⁸, a następnie wywołaj kompilator (`javac`⁹) oraz Wirtualną Maszynę Javy (`java`). Poprawne uruchomienie programów powinno skutkować wyświetleniem listy dostępnych parametrów. Odszukaj parametr odpowiedzialny za wyświetlenie numeru wersji, a następnie uruchom programy ponownie wraz z tym parametrem. Jakie wersje wymienionych programów zostały zainstalowane?

Zadanie 2 – PierwszyProgramKonsola.java

Napisz, skompiluj oraz uruchom program wyświetlający na ekranie napis "Uniwersytet Ekonomiczny w Krakowie". Wykorzystaj dowolny edytor znakowy oraz polecenia wydawane z konsoli. Umieść w pliku źródłowym informacje o autorze programu.

Zadanie 3 – Śledzenie procesu kompilacji

Dokonaj kompilacji programu `PierwszyProgramKonsola.java` stosując parametr `verbose`. Jakie komunikaty generuje kompilator?

Zadanie 4 – Lokalizacja plików binarnych

W katalogu, w którym znajduje się plik źródłowy `PierwszyProgramKonsola.java` utwórz nowy folder o nazwie `Programy`. Następnie dokonaj kompilacji programu `PierwszyProgramKonsola.java` w taki sposób, aby plik wykonywalny `PierwszyProgramKonsola.class` został utworzony w katalogu `Programy`.

Rozwiązanie

Zapoznaj się z parametrami kompilatora. Uruchom proces kompilacji stosując odpowiedni parametr.

Zadanie 5 – PierwszyProgramIDE.java

Wykonaj polecenia zawarte w zadaniu `PierwszyProgramKonsola.java`. Wykorzystaj dowolne IDE.

Zadanie 6 – Choinka.java

Metoda `println()` umożliwia wyprowadzanie danych na konsolę. Napisz program wyświetlający na ekranie poniższą choinkę.

⁸ W systemie operacyjnym Windows (2000, XP, Vista) z menu `Start` wybierz opcję `Uruchom`, następnie wpisz `cmd` i naciśnij klawisz `Enter`.

⁹ Brak dostępu do kompilatora z trybu wiersza poleceń może być związany z brakiem lub nieprawidłowym zdefiniowaniem odpowiedniej ścieżki w zmiennej środowiskowej `PATH`.

```

      *
    ***
  *****
*****
|

```

Zadanie 7 – Inicjaly.java

Napisz program wyświetlający na ekranie własne inicjały (pierwsze litery imienia i nazwiska). Przykładowy rezultat podano poniżej.

```

***  *
*    * *
***** *
*    * *
*  * *****

```

Zadanie 8 – Kot.java

Poniższy program nie kompiluje się. Popraw błędy oraz wygląd kodu źródłowego.

```

//
  Kot
  autor: Piotr Nowak (c) 2009
*/

public clas kot {
  public void Main(String[] args)
    / Kocie myślenie
    system.out.Println("Mruczę, więc jestem... ")
}
}

```

Zadanie 9 – WlasciwosciSrodowiska.java

Poniższy program wyświetla na konsoli numer wersji JRE. Zapoznaj się z dokumentacją klasy `java.lang.System`, a następnie uzupełnij program, aby wyświetlał na ekranie dodatkowo wersję JVM, nazwę katalogu instalacyjnego Javy, nazwę i numer wersji systemu operacyjnego, nazwę użytkownika, nazwę katalogu bieżącego oraz znak końca linii.

```

public class WlasciwosciSrodowiska {
  public static void main(String[] args) {
    System.out.println("Wersja JRE:" + System.getProperty("java.version"));
  }
}

```

Zadanie 10 – LiczbaPseudolosowa.java

Zapoznaj się z dokumentacją klasy `java.lang.Math`. Znajdź metodę generującą liczbę pseudolosową z zakresu $<0,1$). Napisz program wyświetlający taką liczbę. Uruchom program kilka razy. Jakie uzyskujesz wyniki?

Zadanie 11

Odszukaj w witrynie <http://java.sun.com> dowolny kod programu. Skopiuj go na swój nośnik pamięci. Dokonaj jego kompilacji, a następnie uruchom utworzony program.

Rozdział 2

Typy danych, zmienne i operatory

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- efektywnie wykorzystywać dostępne typy prymitywne,
- deklarować zmienne i nadawać im wartości początkowe,
- wykonywać proste operacje arytmetyczne i logiczne,
- odczytywać oraz wyprowadzać dane na konsolę,
- operować złożonymi strukturami danych (tablice).

Wprowadzenie do zagadnień

Program komputerowy składa się z sekwencji instrukcji, które wykonywane są przez komputer dla realizacji zadania. W skład instrukcji wchodzi wyrażenia operujące na danych, reprezentowanych przez zmienne oraz literały. Rodzaj realizowanej operacji określony jest poprzez użycie operatorów.

TYPY DANYCH

Typ danych określa rodzaj informacji oraz zakres dopuszczalnych wartości, jakie może przyjmować stała, zmienna, argument wyrażenia, parametr, czy rezultat metody. Język Java definiuje typy proste (prymitywne) oraz typy złożone (obiektywne). W przypadku tych pierwszych wyróżnić można¹⁰:

- całkowite (`int`, `short`, `long`, `byte`),
- zmiennoprzecinkowe (`float`, `double`),
- znakowy (`char`),
- logiczny (`boolean`),
- pusty (`void`).

Rodzaj informacji oraz zakresy dopuszczalnych wartości określa specyfikacja języka. Warto nadmienić, iż sposób reprezentacji danych jest niezależny od platformy systemowej, a co za tym idzie, zakres dopuszczalnych wartości pozostaje niezmienny niezależnie od platformy, na której uruchamiany jest program.

LITERAŁY

Literały to ciągi znaków reprezentujące wartości zapisane bezpośrednio w kodzie programu. Należy zwrócić uwagę na stosowanie dodatkowej symboliki w celu precyzyjnego określenia typu literału. Poniżej wymienione zostały przykładowe wartości:

¹⁰ Pełna specyfikacja występujących w języku programowania typów danych dostępna jest pod adresem: <http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html>

```

"bordowy"    // łańcuch znaków
'K'          // wartość typu "char"
'\u0041'     // wartość typu "char" w standardzie "unicode" (przedrostek \u)
293          // wartość dziesiętna typu "integer"
49.6         // wartość rzeczywista typu "double"
0251         // wartość ósemkowa (przedrostek 0) typu "integer"
0x25         // wartość szesnastkowa (przedrostek 0x) typu "integer"
400L         // wartość typu "long" (przyrostek L)
2.45F        // wartość typu "float" (przyrostek F lub f)
3.06D        // wartość typu "double" (przyrostek D lub d – domyślny)
9.72E4       // zapis wykładniczy (symbol E lub e)
true         // wartość logiczna

```

Należy zwrócić uwagę na stosowanie dodatkowej symboliki w celu precyzyjnego określenia typu literału (293L, 293F).

Język Java zawiera również zbiór znaków specjalnych (ang. *escape characters*): \b, \t, \f, \r, \n, \", \', \\, możliwych do zastosowania bezpośrednio w kodzie programu:

```
System.out.println("Uniwersytet Ekonomiczny\nw Krakowie");
```

ZMIENNE

Zmienna¹¹ określa obszar pamięci, przeznaczony do przechowywania danych. Każda zmienna posiada nazwę oraz typ, który określa zakres dopuszczalnych wartości oraz rodzaj możliwych do przeprowadzenia operacji. Zmienne prymitywne określają obszar pamięci przechowujący dane typu podstawowego. Natomiast zmienne obiektowe, o których będzie mowa w kolejnych rozdziałach, zawierają referencje (wskazania) do utworzonych wcześniej obiektów. Język Java wymaga deklaracji (określenia nazwy oraz typu) każdej zmiennej przed jej pierwszym użyciem. Pierwszy odczyt wartości zmiennej powinien być poprzedzony jej inicjalizacją, czyli przypisaniem wartości początkowej. Poniżej zamieszczone zostały przykłady ich użycia.

```

int liczbaPojazdow;           // deklaracja (określenie nazwy i typu zmiennej)
double rataKredytu;           // deklaracja
liczbaPojazdow = 4;           // inicjalizacja (nadanie wartości zmiennej)
rataKredytu = 330.58;          // inicjalizacja
double stopaProcentowa = 0.04; // jednoczesna deklaracja i inicjalizacja

```

STAŁE

Podobnie jak zmienna, stała określa obszar pamięci operacyjnej, służący do przechowywania danych. Posiada nazwę oraz typ dopuszczalnych wartości, jakie może przyjmować. W przypadku stałej nie jest możliwa zmiana już raz nadanej wartości w trakcie działania programu. Deklaracja stałej poprzedzona jest słowem kluczowym `final`:

```

final double KURS_EURO = 2.48;
final String NAZWA_UCZELNI = "Uniwersytet Ekonomiczny";
final String MIEJSCOWOSC = "Kraków";
System.out.println(NAZWA_UCZELNI + " " + MIEJSCOWOSC);

```

OPERATORY

Operatory¹² to zestaw symboli w języku programowania określających rodzaj wykonywanej operacji na argumencie lub argumentach wyrażenia. Dokonując klasyfikacji ich podziału, wyróżnić można operatory:

- przypisania: `=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=` `>>>=`
- arytmetyczne¹³: `+` `-` `*` `/` `%`
- jednoargumentowe: `+` `-` `++` `--` `!`
- relacji: `==` `!=` `>` `>=` `<` `<=` `instanceof`
- warunkowe: `&&` `||` `?:`

¹¹ Java wyróżnia 4 kategorie zmiennych: obiektowe (ang. *instance variables*) – unikalne dla każdej instancji klasy, klasowe (ang. *class variables*) – jednakowe dla wszystkich instancji klasy, lokalne (ang. *local variables*) – służące do tymczasowego przechowywania wartości oraz parametry, zadaniem których jest przekazywanie wartości do metody.

¹² Wykaz dopuszczalnych operatorów dostępny jest pod adresem:

<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/operators.html>

¹³ Operator dzielenia daje w wyniku wartość całkowitą (bez reszty), jeśli obydwa argumenty są liczbami całkowitymi. W pozostałych przypadkach wartością jest liczba rzeczywista.

- bitowe: ~ << >> >>> & ^ |

Należy zwrócić uwagę, iż sposób użycia operatorów zależy od typu argumentu (argumentów) natomiast kolejność wykonywania operacji wynika z priorytetu, jaki posiadają operatory. Zmiana kolejności wykonywania operacji realizowana jest przez zastosowanie nawiasów. Poniższe wyrażenia przedstawiają przykładowe wykorzystanie operatorów:

```
komputerPrzenosny = true;
komputerStacjonarny = !komputerPrzenosny;
dystans -= 24;
i++;
iloczynSumy = (1+2)*(3+4);
delta = b*b - 4*a*c;
poleKola = 3.14 * r * r;
sredniaOcen = (ocenaBiologia + ocenaFizyka + ocenaHistoria) / 3;
```

KONWERSJA I RZUTOWANIE TYPÓW

Jeśli argumenty wyrażenia nie są zgodnych typów, zachodzi konieczność dokonania konwersji typu. Gdy przebiega ona bez utraty informacji, wykonywana jest niejawnie. Jeśli może prowadzić do utraty informacji konieczne jest wykonanie rzutowania jawnego:

```
byte x = 17;
int y = x;           //niezgodność typów (konwersja niejawna)
double a = 15.4;
int b = (int) a;      //niezgodność typów (możliwa utrata informacji,
                     //wymagane rzutowanie)
```

ŁAŃCUCHY ZNAKÓW

Łańcuch to ciąg (sekwencja) znaków zawarty w obiekcie lub literale typu `String`¹⁴:

```
String osoba = "Jan Kowalski"; // deklaracja oraz inicjalizacja zmiennej typu String
```

Określenie liczby znaków zmiennej lub literału typu `String` możliwe jest dzięki metodzie `length()`:

```
String nazwaProduktu = "Telefon komórkowy Sony-Ericsson K800i";
System.out.println("Liczba znaków: " + nazwaProduktu.length());
System.out.println("Liczba znaków: " + "Programowanie rozproszone".length());
```

Klasa `java.lang.String` zawiera metody umożliwiające manipulowanie znakami w łańcuchu, porównywanie łańcuchów¹⁵, przeszukiwanie, wyodrębnianie podciągu, tworzenie kopii znaków, zamianę znaków na wielkie/małe litery w oparciu o standard „unicode”, czy konwersję wartości numerycznych do postaci ciągu znaków.

```
String rezultat;
String adres = "ul.Szeroka";
rezultat = "ul.Kwiatowa".equals("ul.Kwiatowa") ? "adres zgodny" : "adres inny";
System.out.println(rezultat);
rezultat = "ul.Kwiatowa".equals(adres) ? "adres zgodny" : "adres inny";
System.out.println(rezultat);
```

TABLICE

Tablice reprezentują kontener służący do przechowywania dowolnej liczby wartości pojedynczego typu. Liczba elementów tablicy jest określona w momencie jej tworzenia i nie może zostać zmieniona w trakcie dalszego działania programu. Każdy element posiada kolejny numer (indeks)¹⁶ umożliwiający zapis i odczyt danych:

¹⁴ Możliwe jest również utworzenie obiektu `String` przy użyciu operatora `new` (zobacz dostępne konstruktory klasy `String`).

¹⁵ Ze względu na sposób reprezentacji danych zaleca się użycie metody `equals()` do porównywania łańcuchów znaków w miejsce operatora porównania `==`.

¹⁶ Numeracja elementów tablicy rozpoczyna się od indeksu 0.

```

/* tablice jednowymiarowe */
// deklaracja zmiennej tablicowej
double[] cenaArtykulu; // deklaracja zmiennej tablicowej (nazwa i typ elementów)
String[] cechaCharakteru; // deklaracja zmiennej tablicowej
int[] rzutKostka;

// utworzenie tablicy (określenie liczby jej elementów)
cenaArtykulu = new double[50];
cechaCharakteru = new String[3];
rzutKostka = new int[8];

// jednoczesna deklaracja zmiennej tablicowej oraz utworzenie tablicy
String[] srodekTransportu = new String[2];

// jednoczesna deklaracja zmiennej, utworzenie tablicy 3-elem. oraz nadanie wartości
String[] sygnalizator = {"czerwony", "żółty", "zielony"};

// dostęp do elementów tablicy (przypisanie i odczyt wartości)
cenaArtykulu[17] = 328.50;
cenaArtykulu[25] = 29.0;
double razem = cenaArtykulu[17] + cenaArtykulu[25];
rzutKostka[7] = 3;
cechaCharakteru[0] = "komunikatywność";
cechaCharakteru[2] = "asertywność";
srodekTransportu[1] = "rower";
System.out.println(sygnalizator[2]);

/* tablice wielowymiarowe */
// deklaracja oraz inicjalizacja tablicy dwuwymiarowej
int[][] tabliczkaMnozenia = {{1,2,3},{2,4,6},{3,6,9}};
System.out.println(tabliczkaMnozenia[2][0]);

```

W celu określenia liczby elementów tablicy należy odczytać wartość pola `length`:

```

String[] koloryTeczy =
    {"czerwony", "pomarańczowy", "żółty", "zielony", "niebieski", "fioletowy"};
int liczbaKolorowTeczy = koloryTeczy.length;

```

Kopiowanie danych pomiędzy tablicami możliwe jest dzięki dostępnej metodzie `java.lang.System.arraycopy()`. Istnieje pokaźna liczba metod umożliwiających manipulowanie na tablicach, realizujących operacje sortowania, czy przeszukiwania elementów. Metody te zostały zgrupowane w klasie `java.util.Arrays`.

TYP WYLICZENIOWY

Typ, składający się z ustalonego, skończonego zbioru stałych wartości (egzemplarzy typu). Każdy z egzemplarzy pozostaje niezmienny w trakcie działania programu. Stąd, zgodnie z konwencją, identyfikatory egzemplarzy powinno tworzyć się przy pomocy wielkich liter i znaku podkreślenia.

```

enum Dzień {PN, WT, SR, CZ, PT, SB, ND};
Dzień dzienPracy = Dzień.WT;
Dzień dzienWolny = Dzień.SB;
System.out.println(dzienPracy!=dzienWolny);

```

Pytania sprawdzające

1. Wskaż różnice, jakie występują pomiędzy zmiennymi, stałymi i literałami.
2. Jakie funkcje w programie pełni deklaracja zmiennej?
3. Na którym z dostępnych typów prymitywnych nie jest możliwe przeprowadzenie operacji rzutowania?
4. Uszereguj operatory według ich priorytetu:

+ % -- <

5. Jaką funkcjonalność realizują operatory: ++ ? : /= % ? Podaj przykłady wyrażeń z ich użyciem.
6. Określ wartość wyrażenia:


```
2 > 3 || 4 < 5 ? 6 << 7 : 8 / 9
```

7. Wymień dopuszczalne wartości (minimalne i maksymalne) dla każdego z typów prymitywnych.
8. Podaj typ wartości wyrażenia: `12 / 7`
9. Które z wymienionych instrukcji nie spowodują powstania błędu kompilacji?

```
float f = 1.01; int i = 1/3; double d = 777d;
```

10. Jaka jest wartość poniższej nierówności? Uzasadnij odpowiedź.

```
0xF3 > 0574
```

11. Określ kody w standardzie unicode dla wymienionych znaków: B 9 Ć # ś
12. Jakie oznaczenie (indeks) posiada pierwszy element tablicy?
13. Zapoznaj się z dokumentacją klasy `java.lang.String`. Odszukaj metodę umożliwiającą łączenie ciągu znaków. Podaj przykład instrukcji łączącej 2 literały zawierające Twoje imię i nazwisko.
14. Jaką informację zwróci wyrażenie `tablica.length` w przypadku tablicy wielowymiarowej?
15. Podaj przykład użycia metody porządkującej elementy tablicy typu `int`.

Zadania do wykonania

PODSTAWOWE OPERACJE NA ZMIENNYCH

Zadanie 12 – OperacjePodstawowe.java

Skompiluj i uruchom poniższy program. Zaobserwuj, jakie informacje wyświetlane są na konsoli. Porównaj wyniki z kodem programu. Czy każda linia kodu jest dla Ciebie zrozumiała?

```
public class OperacjePodstawowe {
    public static void main(String[] args){
        int i = 7;
        System.out.println("Powieść \"Wojna i pokój\" Lwa Tołstoja");
        System.out.println("64kB to " + 0xFFFF + " bitów");
        System.out.println("Czy 32 > 15 ? " + (32>15 ? "tak" : "nie") );
        System.out.println("\u0042 (unicode) reprezentuje znak \u0042");
        System.out.println("7 * (2 do potęgi 3) = " + (7 << 3) );
        System.out.println("Część całkowita 2.56 wynosi " + (int) 2.56);
        System.out.println("7/2 = " + 7/2 + ", ale dlaczego?");
        System.out.println("A dlaczego 7/2d = " + 7/2d + " ?");
        System.out.println("Reszta z dzielenia 10/6 = " + 10%6);
        System.out.println("Wartość i=" + i + ", a ++i=" + (++i));
        System.out.println("Gdy teraz wykonamy operację i*=9 to i=" + (i*=9));
    }
}
```

Zadanie 13 – InstrukcjaPrzypisania.java

Poniższy kod programu zawiera instrukcje przypisania nadające wartości zmiennym, które nie zostały uprzednio zadeklarowane. Skompiluj i uruchom poniższy kod programu. Zaobserwuj, jakie komunikaty o błędach pojawiają się w momencie kompilacji. Następnie uzupełnij program, dodając brakujące deklaracje zmiennych.

```

public class InstrukcjaPrzypisania {
    public static void main(String[] args){
        nazwaPlanety = "Ziemia";
        powierzchniaPlanety = 510065284; // km2
        obwodPlanety = 40041; // km
        albedoPlanety = 0.39;
        planetaZamieszkalna = true;
    }
}

```

Zadanie 14 – Operatory.java

Dokonaj analizy poniższego kodu programu i odpowiedz na pytanie: jakie rezultaty zostaną wyświetlone na konsoli? Następnie uruchom program i porównaj wyniki.

```

public class Operatory {
    public static void main(String[] args){
        int a = 5;
        int b = 2;
        System.out.println("a = " + a + ", b = " + b);
        System.out.println("a/b = " + (a/b) );
        System.out.println("a!=b = " + (a!=b) );
        System.out.println("a%b > --b ? -3 : 17 = " + (a%b > --b ? -3 : 17) );
        System.out.println("++b-a = " + (++b-a) );
        System.out.println("a/(float)b == a/b = " + (a/(float)b == a/b) );
    }
}

```

Zadanie 15 – TypyZmiennych.java

Dla każdego wymienionego literału zadeklaruj zmienną, której typ będzie zgodny z typem literału. Przypisz zmiennej wartość literału. Wyświetl wartości zmiennych na konsoli.

```
493L, false, 3.7, 0x3F, 'B', 0xF7FF, 07246, 1.2e5f, '\u0042', -178609
```

Zadanie 16 – Konwencje.java

Zapoznaj się z konwencją stosowaną podczas pisania kodu źródłowego programu w Javie. Następnie dokonaj stosownych modyfikacji poniższego kodu programu tak, aby jego postać była zgodna z obowiązującymi zasadami.

```

import java.util.Scanner;

public class Konwencje {
    public static void main (String[] args) {
        final double stawkaVat = 1.22;
        double KwotaNetto, KwotaBrutto;

        Scanner SK = new Scanner(System.in);

        System.out.print ("Podaj kwotę netto: ");
        KwotaNetto = SK.nextDouble();
        KwotaBrutto = KwotaNetto * stawkaVat;
        System.out.println ("Kwota Netto: " + KwotaNetto + " Kwota brutto: " +
            KwotaBrutto);
    }
}

```

Zadanie 17 – NiepoprawnyKodProgramu.java

Popraw poniższy kod programu, aby zarówno proces kompilacji, jak i jego wykonania przebiegał bezbłędnie.

```

public class NiepoprawnyKodProgramu {
    public static void main(String[] args){
        String nazwaOperacji = 'Obliczanie podatku VAT';
        int stawkaVAT;
        int podatekVAT;
        int kwotaBrutto;

        kwotaNetto = 231,74;
        stawkaVAT = 22%;
        podatekVAT = kwotaNetto * stawkaVAT;
        KwotaBrutto = KwotaNetto + PodatekVAT;

        System.out.println(nazwaOperacji);
        System.out.println("-----");
        System.out.println("Kwota netto   = " + kwotaNetto);
        System.out.println("Stawka VAT   = " + stawkaVAT);
        System.out.println("Podatek VAT  = " + podatek);
        System.out.println("Kwota brutto = " + kwotaBrutto);
    }
}

```

Zadanie 18 – JednostkiSI.java

Elementy tablicy zawierają wykaz jednostek podstawowych układu SI. Wyświetl na konsoli jednostki temperatury, masy i długości.

Rozwiązanie

```

public class JednostkiSI {
    public static void main(String[] args){
        String[] jednostki = {"kg", "cd", "s", "A", "K", "mol", "m"};
        System.out.println("Jednostki podstawowe układu SI");
        System.out.println("=====");
        System.out.println("Jednostka temperatury (kelvin): " + jednostki[4]);
        System.out.println("Jednostka masy (kilogram): " + jednostki[0]);
        System.out.println("Jednostka długości (metr): " + jednostki[jednostki.length-1]);
        System.out.println("-----");
        System.out.println("Liczba jednostek: " + jednostki.length);
    }
}

```

Zadanie 19 – DniTygodnia.java

Dokonaj modyfikacji poniższego kodu programu, aby odwoływał się do nieistniejącego elementu tablicy. Zaobserwuj pojawiające się komunikaty o błędach.

```

public class DniTygodnia {
    public static void main(String[] args){
        String[] dniTygodnia = {"Poniedziałek", "Wtorek", "Środa", "Czwartek",
        "Piątek", "Sobota", "Niedziela"};

        System.out.println ("Pierwszy dzień tygodnia to: " + dniTygodnia[0]);
        System.out.println ("Dni wolne od pracy to: " + dniTygodnia[5] + ", " +
        dniTygodnia[6]);
    }
}

```

Zadanie 20 – StandardUnicode.java

Wykorzystując wyłącznie notację \u... (standard unicode¹⁷) wyświetl na konsoli swoje imię i nazwisko.

Zadanie 21 – FormatowanieRezultatow.java

Dokonaj formatowania rezultatów wyświetlanych na konsoli.

Rozwiązanie

Poniższy kod programu prezentuje przykładowe sposoby formatowania¹⁸, zarówno ciągu znaków, jak i wartości numerycznych w oparciu o klasę `java.text.MessageFormat`. Zapoznaj się z opisem klasy, a następnie dostosuj sposób wyświetlania daty do formatu `rrrr-mm-dd`.

¹⁷ Opis standardu kodowania znaków dostępny jest pod adresem <http://www.unicode.org> oraz <http://pl.wikipedia.org/wiki/Unicode>

¹⁸ Osoby preferujące styl formatowania dostępny w językach C oraz C++ (funkcja `printf`), mogą skorzystać z narzędzi zawartych w klasie `java.util.Formatter`.

```

import java.text.MessageFormat;
import java.util.Date;

public class FormatowanieRezultatow {
    public static void main(String[] args){
        String s;
        Date teraz = new Date();
        String miejsce = "Kraków";
        int mila = 1852;
        double vat = 0.22;
        double cena = 31560.76;

        s = MessageFormat.format("{0}, {1,date,full}, godz. {1,time,short}",
            miejsce, teraz);
        System.out.println(s);
        s = MessageFormat.format("{0}kB to {1}B", 256,256*1024);
        System.out.println(s);
        s = MessageFormat.format("{0} x {0} x {0} = {1}", 7, 7*7*7);
        System.out.println(s);
        s = MessageFormat.format("{0,number,currency} + VAT({1,number,percent})",
            cena, vat);
        System.out.println(s);
        s = MessageFormat.format("{0} {1} to w przybliżeniu {2}m", 1,
            "mila morska", mila);
        System.out.println(s);
    }
}

```

Zadanie 22 – *FormatowanieAlternatywne.java*

Wykonaj ponownie poprzednie zadanie, wykorzystując jednakże alternatywny sposób formatowania danych, zgodny z funkcją `printf` języka C/C++. Szczegółowy opis znajdziesz w dokumentacji Java API klasy `java.util.Formatter`. Poniżej zostały przedstawione przykłady użycia dostępnych metod:

```

Date teraz = new Date();
System.out.printf("Tu %s - jest godzina %tR\n", "Radio Kraków", teraz);
String tekst = String.format("%s %s", "Java", "Virtual Machine");
System.out.println(tekst);

```

PARAMETRY WIERSZA POLECEŃ

Zadanie 23 – *ParametryWierszaPolecen.java*

Kolejne parametry wiersza poleceń zawierają nazwisko, imię oraz adres zamieszkania. Wyświetl dane na konsoli.

Rozwiązanie

Wartości parametrów wiersza poleceń podane w momencie wywołania programu dostępne są w tablicy `args`, przekazanej jako parametr w metodzie `main()`.

```

/*
 * Przykładowe wywołanie programu:
 * java ParametryWierszaPolecen Kowalewski Eugeniusz "ul. Szeroka 15 m.3"
 */
public class ParametryWierszaPolecen {
    public static void main(String[] args){
        System.out.println("nazwisko: " + args[0]);
        System.out.println("imie: " + args[1]);
        System.out.println("adres: " + args[2]);
    }
}

```

Zadanie 24 – *OperacjeArytmetyczne.java*

Parametry wiersza poleceń zawierają 2 dowolne liczby całkowite, różne od 0. Wyznacz ich sumę.

Rozwiązanie

Poniższy kod programu stanowi rozwiązanie zadania. Dokonaj jego modyfikacji, aby program wyznaczał także różnicę, iloczyn, iloraz oraz resztę z dzielenia dla przekazanych w wierszu poleceń argumentów liczbowych.

```

/*
 * Przykładowe wywołanie programu:
 * java OperacjeArytmetyczne 523 -68
 */

public class OperacjeArytmetyczne {
    public static void main(String[] args){
        // odczyt oraz konwersja liczb do postaci numerycznej
        int liczbaA = Integer.parseInt(args[0]);
        int liczbaB = Integer.parseInt(args[1]);
        // wyświetlenie rezultatów na konsoli
        System.out.print("A = " + liczbaA);
        System.out.println(", B = " + liczbaB);
        System.out.println("A + B = " + (liczbaA+liczbaB));
    }
}

```

Zadanie 25 – KonwerterWalut.java

Napisz program dokonujący zamiany kwoty wyrażonej w dolarach (USD) na euro (EUR). Aktualny kurs wymiany odnajdziesz w sieci Internet. Wartość wyrażoną w USD, jak i bieżący kurs walut przekaż jako parametry wywołania programu.

Zadanie 26 – PodatekVAT.java

W wierszu poleceń umieszczono nazwę towaru oraz jego cenę netto. Wyświetl przekazane informacje na konsoli, a także wartość podatku VAT oraz cenę brutto towaru. Stawkę VAT (jako stałą w programie) przyjmij w wysokości 22%.

ODCZYT DANYCH Z KONSOLI

Zadanie 27 – WielkieLitery.java

Odczytaj z konsoli¹⁹ dowolny ciąg znaków oraz wyświetl go, dokonując zamiany znaków na wielkie litery. Wykorzystaj metody klasy `String`.

Rozwiązanie

```

import java.util.Scanner;

public class WielkieLitery {
    public static void main(String[] args){
        // pobierz tekst z konsoli
        Scanner sc = new Scanner(System.in);
        System.out.print("Wprowadz dowolny tekst: ");
        String tekst = sc.nextLine();
        // wyświetl tekst, dokonując konwersji na wielkie litery
        System.out.println(tekst.toUpperCase());
    }
}

```

Zadanie 28 – DlugoscNapisow.java

Odczytaj z konsoli 2 dowolne ciągi znaków. Wyświetl dłuższy z nich.

Zadanie 29 – FormatRachunku.java

Numer rachunku bankowego składa się z 26 cyfr. Napisz program, który odczyta numer rachunku z konsoli, a następnie wyświetli go w formacie:

```
XX XXXX XXXX XXXX XXXX XXXX XXXX
```

Wykorzystaj dostępne metody klasy `String`.

Zadanie 30 – PeselPlec.java

Numer identyfikacyjny Pesel składa się z 11 cyfr. Dziesiąta z nich określa płeć osoby (cyfra parzysta oznacza kobietę, nieparzysta mężczyznę). Napisz program, który dla dowolnego numeru Pesel wyznaczy płeć osoby. Wyświetl rezultat na konsoli.

¹⁹ Odczyt danych z konsoli może być zrealizowany również innymi metodami, przykładowo przy wykorzystaniu klasy `java.io.Console` (zobacz: <http://java.sun.com/docs/books/tutorial/essential/io/cl.html>).

Zadanie 31 – ZWielkiejLitera.java

Program umożliwia wprowadzanie danych personalnych (imię, nazwisko) pracowników. Wyświetl wprowadzone dane na konsoli w taki sposób, aby imiona i nazwiska rozpoczynały się zawsze od wielkiej litery, przy pozostałych małych, niezależnie od sposobu ich wprowadzenia. Wykorzystaj metody klasy `String`.

Zadanie 32 – KonwerterTemperatur.java

Napisz program, który dla podanej wartości temperatury wyrażonej w stopniach Celsjusza wyznaczy temperaturę w stopniach Fahrenheita oraz Kelvina. Jaka wartość będzie mieć temperatura 30°C wyrażona na tych skalach?

Rozwiązanie

```
import java.util.Scanner;

public class KonwerterTemperatur {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        // odczytaj wartość temperatury w stopniach Celsjusza
        System.out.print("Podaj wartość temperatury w stopniach Celsjusza: ");
        double tempC = sc.nextInt();           // Celsjusz
        // oblicz wartość temperatury na pozostałych skalach
        double tempF = 32 + (9/5d)*tempC;      // Fahrenheit
        double tempK = tempC + 273.15;         // Kelvin
        // wyświetl rezultaty obliczeń, z dokładnością do pełnych stopni
        System.out.println("* temperatura w stopniach C: " + (int)tempC);
        System.out.println("* temperatura w stopniach F: " + (int)tempF);
        System.out.println("* temperatura w stopniach K: " + (int)tempK);
    }
}
```

Zadanie 33 – Wzrost.java

Napisz program, który dla podanej wartości wzrostu wyrażonej w cm, wyświetli na konsoli wzrost wyrażony w stopach i calach.

Zadanie 34 – BMI.java

Napisz program obliczający wskaźnik masy ciała BMI (ang. *Body Mass Index*) na podstawie podanego wzrostu w metrach oraz masy ciała w kg. Formułę wyznaczającą wskaźnik odszukaj w sieci Internet. Wprowadź do programu swój wzrost oraz masę ciała i przekonaj się, czy posiadasz prawidłową wartość wskaźnika BMI.

Rozdział 3

Sterowanie przebiegiem wykonania programu

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- efektywnie sterować przebiegiem wykonania programu,
- łączyć instrukcje programu w grupy,
- tworzyć kod programu wykonywany w zależności od zadanych warunków,
- wielokrotnie wykonywać wybraną instrukcję lub blok instrukcji,
- realizować przy pomocy programu złożone procesy obliczeniowe.

Wprowadzenie do zagadnień

Instrukcje wchodzące w skład programu komputerowego wykonywane są najczęściej sekwencyjnie, począwszy od pierwszej, a skończywszy na ostatniej z nich. Jednakże złożone zadania obliczeniowe wymagają częstokroć łączenia instrukcji w grupy, warunkowego przetwarzania, czy też wielokrotnego ich powtarzania, dla realizacji zadania. Stąd prawie każdy język programowania zaopatrzony jest w zbiór instrukcji sterujących, mających za zadanie umożliwić skonstruowanie programu zgodnie z realizowanymi algorytmem²⁰. Język Java wyróżnia 5 instrukcji sterujących: `while`, `do..while`, `for`, `if`, `switch`, zadaniem których jest umożliwienie rozwiązywania wielorakich problemów, przed którymi stoi twórca programu.

BLOK INSTRUKCJI

Praktyka programowania pokazuje, iż często zachodzi konieczność zbiorczego przetwarzania instrukcji wchodzących w skład realizowanego programu. Taką funkcjonalność zapewnia blok instrukcji. Umożliwia on ich grupowanie, poprzez ujęcie w znaki nawiasów klamrowych `{ }`:

```
{  
    instrukcje;  
}
```

Wewnątrz bloku możliwe jest umieszczenie deklaracji zmiennych. Należy jednak zwrócić uwagę, iż zmienne te widoczne będą wyłącznie wewnątrz tego bloku – nie będzie możliwe odwołanie do zmiennych poza blokiem, w którym zostały zadeklarowane. Również próba deklaracji zmiennej wewnątrz bloku, której nazwa jest identyczna z istniejącą zmienną poza blokiem spowoduje powstanie błędu kompilacji.

²⁰ Algorytm to, w uproszczeniu, skończony uporządkowany ciąg czynności niezbędnych do realizacji zadania.

```

{
    String nrTelefonu = "555 302 197";
    System.out.println(nrTelefonu);
}
System.out.println(nrTelefonu); // zmienna niedostępna - poza blokiem instrukcji

```

INSTRUKCJA WARUNKOWA

Ten rodzaj instrukcji sterującej umożliwia warunkowe wykonanie pojedynczej instrukcji lub bloku instrukcji programu w zależności od wartości wyrażenia logicznego. Jeśli wyrażenie logiczne jest prawdziwe (przyjmuje wartość `true`) to wykonaniu podlegać będą instrukcje programu występujące wewnątrz bloku instrukcji²¹ `if`. W przypadku, gdy wyrażenie logiczne przyjmie wartość `false`, żadna z instrukcji umieszczonych w bloku instrukcji nie zostanie wykonana.

```

if (wyrażenie_logiczne) {
    instrukcje;
}

```

W poniższym przykładzie obydwie instrukcje programu umieszczone w bloku instrukcji zostaną wykonane jedynie w przypadku, gdy wartość zmiennej `x` będzie różna od 0.

```

int x = 5;
if (x != 0) {                // wyrażenie logiczne przyjmuje wartość true
    x *= 3;                  // zatem obydwie instrukcje zostaną wykonane
    y = x + 4;
}

```

Dostępna jest również wersja instrukcji warunkowej `if` uwzględniająca sytuację alternatywną. Dodatkowy blok instrukcji umieszczony po klauzuli `else` zostanie wykonany jedynie wtedy, gdy wyrażenie logiczne przyjmie wartość `false`.

```

if (wyrażenie_logiczne) {
    instrukcje1;
} else {
    instrukcje2;
}

```

Poniższy przykład demonstruje zastosowanie klauzuli `else`. Blok instrukcji występujący po klauzuli `else` zostanie wykonany jedynie wtedy, gdy wartość `x` będzie równa lub mniejsza od 0.

```

int x = 0;
if (x > 0) {                // wyrażenie logiczne przyjmuje wartość false
    x = 2*x+4;              // zatem instrukcja nie zostanie wykonana
} else {
    x = x - 5;              // obydwie instrukcje zostaną wykonane
    y = 2*x;
}

```

W przypadku, gdy liczba warunków podlegających sprawdzeniu jest duża, możliwe jest stosowanie zagnieżdżania instrukcji warunkowych:

```

int x = 5;
if (x > 0) {
    System.out.println("Dodatnia wartość x");
} else if (x < 0) {
    System.out.println("Ujemna wartość x");
} else {
    System.out.println("Wartość x wynosi 0");
}

```

INSTRUKCJA WIELOKROTNEGO WYBORU

W przypadku, gdy liczba warunków podlegających sprawdzeniu jest znaczna, stosowanie instrukcji `if-else` może stać się nieporęczne. W takiej sytuacji możliwe jest użycie instrukcji wielokrotnego wyboru `switch`. Jej działanie polega na wyznaczeniu wartości wyrażenia, a następnie porównaniu tej wartości z

²¹ Gdy instrukcja warunkowa `if` zawiera wyłącznie jedną instrukcję programu, wtedy stosowanie bloku instrukcji (użycie pary nawiasów klamrowych `{ }`) nie jest wymagane.

wymienionymi po klauzulach `case` wartościami stałych. W przypadku równości sterowanie przekazywane jest do instrukcji występującej po klauzuli `case`, dla której wartość stałej jest zgodna z wartością wyrażenia. Jeśli żadna równość nie zachodzi, wykonywane są instrukcje występujące po słowie kluczowym `default` lub kolejne instrukcje programu, gdy klauzula ta nie została użyta. Instrukcja `break` umożliwia przerwanie wykonywania instrukcji `switch` i przeniesienie sterowania do kolejnych instrukcji programu występujących po instrukcji `switch`.

```
switch (wyrażenie) {
    case wartość1:
        instrukcje1;
        break;
    case wartość2:
        instrukcje2;
        break;
    case wartośćN:
        instrukcjeN;
        break;
    default:           // część opcjonalna instrukcji switch
        instrukcje;
}
```

Poniższy przykład ilustruje użycie instrukcji wielokrotnego wyboru. W zależności od wartości, jaką przyjmuje zmienna `poraRoku`, na konsolę zostanie wyprowadzony stosowny napis.

```
char poraRoku = 'Z';
switch(poraRoku) {
    case 'W':
        System.out.println("wiosna");
        break;
    case 'L':
        System.out.println("lato");
        break;
    case 'J':
        System.out.println("jesień");
        break;
    case 'Z':
        System.out.println("zima");           // wartość stałej zgodna z wartością wyrażenia
        break;                               // instrukcja zostanie wykonana
    default:
        System.out.println("symbol nieprawidłowy");
}
```

PĘTLE NIEOKREŚLONE

Instrukcje iteracyjne (pętle) umożliwiają wielokrotne wykonywanie pojedynczej instrukcji lub bloku instrukcji. W przypadku, gdy nie jest z góry znana liczba powtórzeń, stosowane są pętle nieokreślone. Jedną z nich jest instrukcja `while`, której działanie polega na wyznaczeniu wartości wyrażenia logicznego. Jeśli wyrażenie to przyjmuje wartość prawdy (`true`) wykonywany jest blok instrukcji występujący po wyrażeniu, a następnie ponownie obliczana jest wartość wyrażenia logicznego. Blok instrukcji będzie wykonywany wielokrotnie do momentu, gdy wyrażenie logiczne nie przyjmie wartości `false`. Wtedy sterowanie zostanie przekazane do kolejnej instrukcji programu, występującej po instrukcji `while`.

```
while (wyrażenie_logiczne) {
    instrukcje;
}
```

Poniższy kod programu przedstawia przykład użycia instrukcji iteracyjnej `while`. Blok instrukcji programu będzie wykonywany do momentu, aż zmienna `x` nie przyjmie wartości mniejszej od 0 (w tym przypadku blok instrukcji zostanie wykonany ośmiokrotnie). Należy zwrócić uwagę, iż zwykle konieczna jest modyfikacja wartości wyrażenia logicznego wewnątrz bloku instrukcji, aby możliwe było zakończenie wykonywania instrukcji `while` w skończonej liczbie powtórzeń.

```
int x = 7;
while (x >= 0) {
    x--;
    System.out.println(x);
} // obydwie instrukcje wchodzące w skład bloku instrukcji
// zostaną wykonane ośmiokrotnie
```

Odmianą instrukcji `while` jest pętla nieokreślona `do..while`, której działanie jest zbliżone. Jedyna różnica polega na wyznaczaniu wartości wyrażenia logicznego dopiero po wykonaniu bloku instrukcji. Zatem, w przeciwieństwie do instrukcji `while`, blok instrukcji zostanie wykonany przynajmniej jeden raz.

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

PĘTLE OKREŚLONE

Jeśli znana jest z góry liczba powtórzeń, możliwe jest użycie instrukcji iteracyjnej `for`. Jej działanie polega na ustaleniu wartości początkowej, określającej początek iteracji. Wykonanie bloku instrukcji jest uwarunkowane wartością wyrażenia logicznego. Jeśli jest ono prawdziwe (wartość `true`) blok instrukcji jest wykonywany, a następnie wykonywane jest wyrażenie modyfikujące ustaloną wartość początkową. W kolejnym kroku ponownie sprawdzana jest wartość wyrażenia logicznego i w przypadku wartości `true` wykonywany blok instrukcji programu. Opisane czynności są powtarzane do momentu, gdy wyrażenie logiczne przyjmie wartość `false`. Wtedy sterowanie programu przekazane zostaje do kolejnych instrukcji występujących po instrukcji `for`.

```
for (wartość_początkowa; wyrażenie_logiczne; wyrażenie_modyfikujące) {
    instrukcje;
}
```

W poniższym przykładzie blok instrukcji programu występujący w instrukcji iteracyjnej `for` zostanie wykonany ośmiokrotnie, wyświetlając każdorazowo na konsoli wartość zmiennej `i`.

```
for (int i=5; i<=12; i++) {
    System.out.print("Wartość i=");
    System.out.println(i);
}
```

Jedną z odmian instrukcji `for` jest konstrukcja programowa, umożliwiająca wykonanie bloku instrukcji programu dla każdego elementu tablicy²². W każdej iteracji zmienna `element` przyjmuje wartość kolejnego przetwarzanego elementu tablicy. Liczba iteracji jest równa w tym wypadku liczbie elementów przetwarzanej tablicy.

```
for (element : tablica) {
    instrukcje
}
```

Poniższy przykład wyświetla na konsoli w oddzielnych wierszach wszystkie wartości elementów tablicy `poraDnia`.

```
String[] poraDnia = {"świt", "zmierzch", "mrok"};
for (String pora : poraDnia) {
    System.out.println(pora);
}
```

PRZERYWANIE DZIAŁANIA INSTRUKCJI STEROWANIA

Język Java umożliwia sterowanie wykonaniem instrukcji iteracyjnych. Instrukcja `break` umieszczona wewnątrz bloku instrukcji powoduje zakończenie wykonywania iteracji i przeniesienie sterowania do instrukcji programu występującej po instrukcji iteracyjnej. Natomiast instrukcja `continue` przerywa bieżącą iterację, przekazując sterowanie do miejsca, gdzie wyznaczana jest wartość wyrażenia logicznego

²² Instrukcji `for` można użyć do wszystkich obiektów stanowiących kolekcję elementów.

kontrolującego wykonywanie pętli. Poniższy przykład z użyciem instrukcji `continue` wyświetla na konsoli wyłącznie cyfry z badanego ciągu znaków (zmienna `nrDowoduOsobistego`).

```
for (int i=0; i<nrDowoduOsobistego.length(); i++){

    /* gdy znak nie jest cyfrą, pobierz kolejny znak */
    if (!Character.isDigit(nrDowoduOsobistego.charAt(i)))
        continue;

    /* wyświetl cyfrę na konsoli */
    System.out.print(nrDowoduOsobistego.charAt(i));
}
```

Pytania sprawdzające

1. Wymień nazwy wszystkich instrukcji sterujących możliwych do użycia w języku Java.
2. Podaj definicję algorytmu. Jakże wyróżniamy sposoby jego zapisu?
3. Przedstaw algorytm wyznaczania silni wartości N przy użyciu pseudokodu.
4. Narysuj lub odszukaj w sieci Internet schemat blokowy reprezentujący algorytm wyznaczania pierwiastków równania kwadratowego postaci $ax^2+bx+c=0$.
5. Czym jest blok instrukcji i jaki jest sposób jego notacji w języku programowania Java?
6. Jaki jest zasięg widoczności zmiennych zadeklarowanych wewnątrz bloku instrukcji?
7. Wymień operatory logiczne możliwe do użycia w instrukcji warunkowej `if`.
8. Jaką rolę w instrukcji warunkowej pełni słowo kluczowe `else`?
9. Jaki będzie rezultat działania poniższego kodu programu?

```
int i = 0;
if (i++ > 0) {
    System.out.println(i);
}
```

10. Które z wymienionych typów danych może przyjmować wyrażenie instrukcji `switch`?

```
boolean byte short int long char float double
```

11. Jaką funkcję pełni klauzula `default` w instrukcji wyboru? Czy jej użycie jest każdorazowo wymagane?
12. Jaki będzie rezultat działania poniższego kodu programu?

```
int i = 2;
switch (i) {
    case 1:
        System.out.println("jeden");
        break;
    case 2:
        System.out.println("dwa");
    case 3:
        System.out.println("trzy");
    default:
        System.out.println("inna liczba");
}
```

13. Wymień i scharakteryzuj składowe instrukcje `for`.
14. Podaj składnię zapisu pętli określonej, zadaniem której będzie wyświetlenie na konsoli wszystkich parzystych liczb naturalnych z przedziału 50..100.
15. Jaki będzie rezultat działania poniższego programu?

```
int x = 5;
for (int i=1; i<8; i+=4) {
    x += i;
}
System.out.println(x);
```

16. Omów zasadę funkcjonowania odmiany instrukcji `for` przeznaczonej do operacji na elementach tablicy.
17. Wymień zasadnicze różnice pomiędzy pętlą określoną `for`, a pętlą nieokreśloną `while`.
18. Jaką funkcję pełnią instrukcje `break` oraz `continue` użyte w instrukcji sterującej?
19. Czym różnią się instrukcje iteracyjne `while` oraz `do..while` ? Wskaż zasadnicze różnice.
20. Wymień instrukcje sterujące, w których możliwe jest użycie instrukcji `break`.

Zadania do wykonania

PRZETWARZANIE WARUNKOWE

Zadanie 35 – *LiczbaParzysta.java*

Sprawdź, czy odczytana z konsoli liczba naturalna jest parzysta.

Rozwiązanie

```
import java.util.Scanner;

public class LiczbaParzysta {
    public static void main(String[] args) {
        int liczbaNaturalna; // analizowana wartość
        /* pobierz liczbę z konsoli */
        Scanner sc = new Scanner(System.in);
        System.out.print("Wprowadź dowolną liczbę naturalną: ");
        liczbaNaturalna = sc.nextInt();
        /* sprawdź, czy liczba jest parzysta (podzielna przez 2 bez reszty) */
        boolean liczbaParzysta = liczbaNaturalna % 2 == 0 ? true : false;
        /* wyświetl rezultat na konsoli */
        if (liczbaParzysta) {
            System.out.printf("Liczba %d jest parzysta", liczbaNaturalna);
        } else {
            System.out.printf("Liczba %d jest nieparzysta", liczbaNaturalna);
        }
    }
}
```

Zadanie 36 – *PodatekDochodowy.java*

Podatek dochodowy jest obowiązkowym świadczeniem obywatela na rzecz państwa. Wartość podatku wyznaczana jest na podstawie osiągniętych dochodów oraz obowiązującej skali podatkowej. Napisz program, który dla wartości dochodu odczytanej z konsoli wyznaczy wartość należnego podatku. Obowiązującą skalę podatkową odszukaj w sieci Internet.

Zadanie 37 – *RównanieKwadratowe.java*

Równanie kwadratowe jest równaniem algebraicznym z jedną niewiadomą postaci $ax^2+bx+c=0$. Napisz program, wyznaczający pierwiastki równania kwadratowego.

Rozwiązanie

Zwróć uwagę na metodę `sqrt()` wyznaczającą wartość pierwiastka kwadratowego (klasa `java.lang.Math`) oraz sposób formatowania rezultatów zadania.

```

import java.util.Scanner;

public class RownanieKwadratowe {
    public static void main(String[] args) {
        double x1, x2; // pierwiastki równania
        // odczytaj współczynniki równania z konsoli
        Scanner sc = new Scanner(System.in);
        System.out.println("Współczynniki równania kwadratowego");
        System.out.println("postaci  ax2 + bx + c = 0");
        System.out.print("a = ");
        double a = sc.nextDouble();
        System.out.print("b = ");
        double b = sc.nextDouble();
        System.out.print("c = ");
        double c = sc.nextDouble();
        // jeden pierwiastek równania
        if( a == 0) {
            x1 = -c/b;
            System.out.printf("Pierwiastek równania: x = %s", x1);
            return;
        }
        // wyznacz wyróżnik równania
        double delta = b*b - 4*a*c;
        // wyznacz wartości pierwiastków równania
        if (delta > 0) {
            x1 = (-b - Math.sqrt(delta))/(2*a);
            x2 = (-b + Math.sqrt(delta))/(2*a);
            System.out.printf("Pierwiastki równania: x1 = %s, x2 = %s", x1, x2);
        } else if (delta == 0) {
            x1 = -b / (2*a);
            System.out.printf("Pierwiastek równania: x = %s", x1);
        } else {
            System.out.println("Brak pierwiastków równania!");
        }
    }
}

```

Zadanie 38 – OcenaSloownie.java

Ocena stanowi umowny sposób zakwalifikowania postępów ucznia lub studenta. Może zostać przedstawiona w zapisie symbolicznym (np. cyfry od 1 do 6) lub słownym. Napisz program, który dla wartości numerycznej oceny odczytanej z konsoli wyświetli jej słowny zapis (celujący, bardzo dobry, dobry, dostateczny, mierny, niedostateczny). Kod programu powinien wykorzystywać instrukcję wielokrotnego wyboru switch.

Rozwiązanie

```

import java.util.Scanner;

public class OcenaSloownie {
    public static void main(String[] args) {
        /* odczytaj dane z konsoli */
        Scanner sc = new Scanner(System.in);
        System.out.print("ocena (1..6): ");
        int n = sc.nextInt();
        /* wyświetl ocenę */
        System.out.print("ocena " + n + " ");
        switch(n) {
            case 1:
                System.out.println("niedostateczny");
                break;
            case 2:
                System.out.println("mierny");
                break;
            case 3:
                System.out.println("dostateczny");
                break;
            case 4:
                System.out.println("dobry");
                break;
            case 5:
                System.out.println("bardzo dobry");
                break;
            case 6:
                System.out.println("celujący");
        }
    }
}

```

```

        break;
    default:
        System.out.println("nieprawidłowa!");
    }
}
}

```

Zadanie 39 – *ProstyKalkulator.java*

Kalkulator stanowi urządzenie elektroniczne służące do wykonywania obliczeń matematycznych. Napisz program symulujący działanie prostego kalkulatora, który dla odczytanych z konsoli wartości dwóch argumentów rzeczywistych oraz jednego z operatorów (+-*/) wyznaczy wartość wyrażenia. Format komunikacji z użytkownikiem przedstawiono poniżej.

```

KALKULATOR
wprowadź argument 1: 17
wprowadź argument 2: 29
wprowadź operator: +
rezultat: 17 + 29 = 46

```

PRZETWARZANIE ITERACYJNE

Zadanie 40 – *Silnia.java*

Silnią liczby naturalnej N nazywamy iloczyn wszystkich liczb naturalnych nie większych niż N . Napisz program wyznaczający wartość silni dla zadanego N , odczytanego z konsoli.

Rozwiązanie

```

import java.util.Scanner;

public class Silnia {
    public static void main(String[] args) {
        long silnia = 1;
        /* odczytaj dane z konsoli */
        Scanner sc = new Scanner(System.in);
        System.out.print("Liczba naturalna (1..20): ");
        int n = sc.nextInt();
        /* oblicz wartość silni */
        for (int i=1; i<=n; i++) {
            silnia *= i;
        }
        /* wyświetl rezultaty */
        System.out.println(n + "! = " + silnia);
    }
}

```

Zadanie 41 – *CiagArytmetyczny.java*

Napisz program wyświetlający na konsoli N początkowych wyrazów ciągu arytmetycznego o różnicy równej 3. Wartość N odczytaj z konsoli. Przykładowy rezultat podano poniżej.

```

Ciąg arytmetyczny o różnicy 3: 1, 4, 7, 10, 13, ...

```

Zadanie 42 – *TabliczkaMnozenia.java*

Stosując pętle określone, napisz program wyświetlający na konsoli tabliczkę mnożenia z zakresu od 1 do 12. Przedstaw rezultat jak pokazano poniżej. Dla sformatowania wyników użyj metody klasy `java.util.Formatter` (metoda `printf()`).

```

1  2  3  4  5  6  7  8  9 10 11 12
2  4  6  8 10 12 14 16 18 20 22 24
3  6  9 12 15 18 21 24 27 30 33 36
4  8 12 16 20 24 28 32 36 40 44 48
5 10 15 20 25 30 35 40 45 50 55 60
6 12 18 24 30 36 42 48 54 60 66 72
7 14 21 28 35 42 49 56 63 70 77 84
8 16 24 32 40 48 56 64 72 80 88 96
9 18 27 36 45 54 63 72 81 90 99 108
10 20 30 40 50 60 70 80 90 100 110 120
11 22 33 44 55 66 77 88 99 110 121 132
12 24 36 48 60 72 84 96 108 120 132 144

```

Zadanie 43 – *LiczbyPierwsze.java*

Liczbę naturalną większą od 1 nazywamy liczbą pierwszą, jeśli ma ona dokładnie 2 dzielniki naturalne o wartościach 1 oraz tej liczby. Napisz program znajdujący N początkowych liczb pierwszych. Wyświetl rezultaty na konsoli w formacie, jak poniżej. Wartość N odczytaj z konsoli.

```
Liczby pierwsze: 2 3 5 7 11 ...
```

Rozwiązanie

Wykorzystując instrukcje iteracyjne sprawdź, czy liczba N jest podzielna jedynie przez 1 oraz przez N.

Zadanie 44 – *PoprawnaLiczbaNaturalna.java*

Jedną z podstawowych funkcji programu jest zapewnienie poprawności wprowadzanych informacji. Napisz program, który sprawdza, czy wprowadzony z konsoli ciąg znaków jest poprawną liczbą naturalną (składa się wyłącznie z cyfr).

Rozwiązanie

Odszukaj w dokumentacji, jaką funkcję pełni metoda `isDigit()` zawarta w klasie `java.lang.Character`. Sprawdź, jakie inne metody zawiera ta klasa.

```

import java.util.Scanner;

public class PoprawnaLiczbaNaturalna {
    public static void main(String[] args) {
        char znak;
        Scanner sc = new Scanner(System.in);
        /* wprowadzenie ciągu znaków */
        System.out.print("liczba naturalna: ");
        String liczba = sc.nextLine();
        /* sprawdzenie, czy każdy znak ciągu jest cyfrą */
        boolean liczbaOK = true;
        for (int i=0; i<liczba.length(); i++) {
            znak = liczba.charAt(i);
            if (!Character.isDigit(znak))
                liczbaOK = false;
        }
        /* podanie rezultatów */
        if (liczbaOK) {
            System.out.printf("ciąg '%s' jest liczbą",liczba);
        } else {
            System.out.printf("ciąg '%s' nie jest liczbą",liczba);
        }
    }
}

```

Zadanie 45 – *PoprawnaLiczbaCalkowita.java*

Dokonaj takiej modyfikacji programu `PoprawnaLiczbaNaturalna.java`, aby weryfikował on, czy wprowadzony ciąg znaków jest poprawną liczbą całkowitą (składa się wyłącznie z cyfr oraz opcjonalnie ze znaku '-' poprzedzającego cyfry).

Zadanie 46 – *Sumator.java*

Wyznacz sumę dowolnej liczby wartości rzeczywistych. Napisz program, który odczytuje kolejne wartości rzeczywiste z konsoli do momentu, gdy użytkownik nie wprowadzi wartości 0. Rezultatem działania programu jest wyznaczenie sumy wprowadzonych wartości.

Zadanie 47 – WeryfikacjaPesel.java

Numer Pesel, składający się dokładnie z 11 cyfr stanowi identyfikator pozycji w rejestrze, w systemie ewidencji ludności Polski. Ostatnia cyfra numeru Pesel jest cyfrą kontrolną, pozwalającą na sprawdzenie poprawności numeru. Napisz program, który pozwoli na sprawdzanie poprawności numerów Pesel.

Rozwiązanie

Należy sprawdzić: długość ciągu (11 znaków), jego zawartość (wyłącznie cyfry) oraz zweryfikować poprawność cyfry kontrolnej. Algorytm sprawdzania poprawności cyfry kontrolnej odszukaj w sieci Internet.

Zadanie 48 – ZdanieWspak.java

Wyświetl na konsoli dowolny ciąg znaków wspak (od znaku ostatniego, do pierwszego).

Rozwiązanie

```
public class ZdanieWspak {
    public static void main(String[] args) {
        String zdanie = "Pierwsze koty za płoty";
        System.out.println("Zdanie: " + zdanie);
        System.out.print("Zdanie wspak: ");
        for (int i=zdanie.length()-1; i>=0; i--) {
            System.out.print(zdanie.charAt(i));
        }
    }
}
```

Zadanie 49 – LiczbaSloownie.java

Napisz program, który wyświetl na konsoli zapis słowny dowolnej liczby naturalnej. Format rezultatu przedstawiono poniżej.

38227 - trzy osiem dwa dwa siedem

Zadanie 50 – Alfabet.java

Wyświetl na konsoli kolejne wielkie litery alfabetu łacińskiego A..Z rozdzielone znakiem odstępu.

Rozwiązanie

Typ `char` jest typem porządkowym (można wyznaczyć poprzedni i następny element), stąd możliwe jest jego użycie w pętli określonej `for`.

```
public class Alfabet {
    public static void main(String[] args) {
        System.out.print("Ciąg wielkich liter alfabetu łacińskiego: ");
        for (char litera='A'; litera <= 'Z'; litera++) {
            System.out.print(litera + " ");
        }
    }
}
```

Zadanie 51 – PismoRozstrzelone.java

Wyświetl na konsoli dowolny ciąg znaków wprowadzony przez użytkownika w formie rozstrzelonej.

Rozwiązanie

Dodaj znak odstępu pomiędzy wszystkimi znakami wyświetlanego ciągu.

Zadanie 52 – AnalizaWyrazow.java

Zapoznaj się z dokumentacją dotyczącą metody `split()` klasy `java.lang.String`. Następnie napisz program, który dowolny tekst odczytany z konsoli podzieli na poszczególne wyrazy, każdy z nich wyświetlając w odrębnej linii.

Zadanie 53 –LiczbyPseudolosoweUjemne.java

Dokonaj modyfikacji zadania `LiczbaPseudolosowa.java` generującego liczbę pseudolosową. Wyświetl na konsoli 20 całkowitych liczb pseudolosowych z przedziału -20..-5.

Zadanie 54 – SymulatorRzutuKostka.java

Napisz program, który zasymuluje N rzutów kostką do gry o liczbie oczek 1..6. Oblicz częstość wyrzucenia poszczególnych liczb oczek. Rezultaty wyświetl na konsoli w formacie, jak przedstawiono poniżej. Wykorzystaj tablice.

```

Rezultat 10 rzutów kostką
=====
Liczba oczek | Liczebność
-----
      1      |      3
      2      |      0
      3      |      1
      4      |      2
      5      |      4
      6      |      0

```

Zadanie 55 – Monety.java

W obiegu są monety 1, 2 i 5 zł. Napisz program przedstawiający dowolną kwotę (liczbę naturalną) za pomocą jak najmniejszej liczby monet.

Rozwiązanie

```

import java.util.Scanner;

public class Monety {
    public static void main(String[] args) {
        int kwota;
        Scanner sc = new Scanner(System.in);
        /* pobierz kwotę z konsoli */
        do {
            System.out.print("Podaj kwotę w zł (1..1000): ");
            kwota = sc.nextInt();
        } while (kwota < 1 || kwota > 1000);
        /* wyznacz najmniejszą możliwą liczbę monet */
        System.out.println("-----");
        System.out.println("monety 5zł: " + kwota/5);
        System.out.println("monety 2zł: " + (kwota%5)/2);
        System.out.println("monety 1zł: " + (kwota%5)%2);
    }
}

```

Zadanie 56 – QuickSort.java

Napisz program, który utworzy, uporządkuje i wyświetli na konsoli dowolną liczbę wartości naturalnych z przedziału [a,b].

Rozwiązanie

W celu uporządkowania wartości skorzystaj z jednej z metod klasy `java.util.Arrays`. Jaką funkcję pełni w poniższym programie metoda `println()` bez parametrów?

```

import java.util.Scanner;
import java.util.Arrays;

public class QuickSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // określ dopuszczalne wartości liczbowe
        System.out.println("Generator n liczb pseudolosowych z przedziału [a,b]");
        System.out.println("=====");
        System.out.print("ilość liczb (n): ");
        int n = sc.nextInt();
        System.out.print("wartość minimalna (a): ");
        int a = sc.nextInt();
        System.out.print("wartość maksymalna (b): ");
        int b = sc.nextInt();
        // utwórz zbiór liczb naturalnych z przedziału [a,b]
        long[] zbiorLiczby = new long[n];
        for (int i=0; i<zbiorLiczby.length; i++) {
            zbiorLiczby[i] = a + (int) (Math.random() * (b-a+1));
        }
        // wyświetl liczby nieuporządkowane
        System.out.printf("liczby nieuporządkowane (%d..%d): ", a, b);
        for (long x : zbiorLiczby) {
            System.out.print(x + " ");
        }
        System.out.println();
        // uporządkuj liczby
        Arrays.sort(zbiorLiczby);
        // wyświetl liczby uporządkowane
        System.out.printf("liczby uporządkowane (%d..%d): ", a, b);
        for (long x : zbiorLiczby) {
            System.out.print(x + " ");
        }
        System.out.println();
    }
}

```

Zadanie 57 – KartotekaPersonalna.java

Napisz program umożliwiający utworzenie kartoteki zawierającej dane personalne pracownika. Wprowadź dane do kartoteki, a następnie wyświetl jej zawartość na konsoli.

Rozwiązanie

Typ wyliczeniowy enum jest w rzeczywistości typem obiektywnym, posiadającym metodę `values()` zwracającą tablicę typu `String`, zawierającą nazwy egzemplarzy typu. Metoda `ordinal()` umożliwia określenie numeru porządkowego egzemplarza (numeracja od 0).

```
import java.util.Scanner;

enum DaneOsobowe {NAZWISKO, IMIE, ADRES, KOD_POCZTOWY, MIEJSCOWOŚĆ};

public class KartotekaPersonalna {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        /* zakładanie kartoteki */
        // metoda values() tworzy tablicę egzemplarzy typu DaneOsobowe
        int liczbaElementowKartoteki = DaneOsobowe.values().length;
        String[] kartoteka = new String[liczbaElementowKartoteki];
        /* wprowadzanie danych do kartoteki */
        System.out.println("Wprowadź dane personalne");
        for (DaneOsobowe poleDanych: DaneOsobowe.values()) {
            System.out.print(poleDanych + ": ");
            kartoteka[poleDanych.ordinal()] = sc.nextLine();
        }
        /* wyświetlanie danych z kartoteki */
        System.out.println("\nDANE PERSONALNE");
        System.out.printf("%s %s, %s %s, %s\n",
            kartoteka[DaneOsobowe.IMIE.ordinal()],
            kartoteka[DaneOsobowe.NAZWISKO.ordinal()],
            kartoteka[DaneOsobowe.KOD_POCZTOWY.ordinal()],
            kartoteka[DaneOsobowe.MIEJSCOWOŚĆ.ordinal()],
            kartoteka[DaneOsobowe.ADRES.ordinal()]);
    }
}
```

Zadanie 58 – *SymulatorLottomatu.java*

„Duży Lotek” jest jedną z gier liczbowych, polegającą na prawidłowym wytypowaniu 6 liczb z 49 możliwych. Klient może wskazać dowolne liczby, może też zdać się na metodę „chybił-trafił”, gdzie maszyna (lottomat) generuje dla niego losową kombinację 6 różnych liczb. Napisz program symulujący funkcję lottomatu. Klient zamawia pewną liczbę zakładów (1..8). Program generuje liczby, uporządkowane rosnąco. Wyniki prezentowane są na konsoli w formacie przedstawionym poniżej.

```
ZAKŁADY DUŻEGO LOTKA
=====
1/  5 12 23 28 45 48
2/ 17 34 35 38 41 44
3/  2  5 14 27 28 49
4/ ...
```

Zadanie 59 – *AlternatywnaTabliczkaMnozenia.java*

Zmodyfikuj kod programu z zadania *TabliczkaMnozenia.java*. Użyj wyłącznie pętli nieokreślonych.

Zadanie 60 – *CiągFibonacciego.java*

Ciąg Fibonacciego to ciąg liczb naturalnych, w którym każdy wyraz ciągu jest sumą dwóch poprzednich (z wyjątkiem wyrazu pierwszego i drugiego):

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ dla } n \geq 2 \end{aligned}$$

Napisz program, który wyznaczy N początkowych wyrazów ciągu Fibonacciego. Wartość N odczytaj z konsoli. Przykładowe początkowe wartości ciągu:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, ...
```

Zadanie 61 – *AlgorytmEuklidesa.java*

Algorytm Euklidesa to metoda wyznaczania największego wspólnego podzielnika (NWD) dla dwóch dowolnych liczb naturalnych. Napisz program wyznaczający wspólny dzielnik dla liczb naturalnych p i q wprowadzonych z konsoli. Opis algorytmu odszukaj w sieci Internet.

Zadanie 62 – *GeneratorHasel.java*

Napisz program generujący 6-cio literowe hasła. Każde z nich składa się dokładnie z trzech spółgłosek, występujących na pozycjach nieparzystych oraz 3 samogłosek występujących na

pozostałych pozycjach wyrazu. Hasło składa się wyłącznie z małych liter alfabetu łacińskiego. Przykładowe hasła podano poniżej.

hulapi, banano, geruna, wileta, ...

Zadanie 63 –SitoEratostenesa.java

Grecki matematyk Eratostenes, żyjący ok. 200 lat p.n.e. podał sposób znajdowania liczb pierwszych znany pod nazwą Sito Eratostenesa. Napisz program znajdujący liczby pierwsze mniejsze bądź równe N. Wartość N odczytaj z konsoli. Opis algorytmu odszukaj w sieci Internet.

Rozdział 4

Tworzenie obiektów i metod

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:


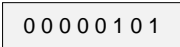
- wskazać różnice pomiędzy zmienną typu prostego, a obiektem (zmienną typu klasowego),
- prawidłowo tworzyć nowe obiekty,
- posługiwać się referencją,
- tworzyć proste metody,
- wykorzystywać standardowe klasy biblioteczne (`Scanner`, `Math`, `String`,...).

Wprowadzenie do zagadnień

Java jest językiem silnie ukierunkowanym na obiektowość. Tworzenie programów w ogólnym zarysie polega na tworzeniu obiektów na bazie klas oraz wykorzystaniu właściwych metod znajdujących się w tych klasach. Poprzednie moduły poruszały problematykę dotyczącą typów prostych (`int`, `float`, czy `boolean`), a także podstawowych klas bibliotecznych (`Scanner`, `String`, `System`, `Math`). Poniżej zostaną omówione zagadnienia prezentujące różnice pomiędzy wymienionymi typami danych, jak również proces tworzenia obiektów oraz metod.

ZMIENNE TYPU PROSTEGO

Deklaracja zmiennej typu prostego jest równoznaczna z rezerwacją obszaru w pamięci komputera przeznaczonego do przechowywania wartości zmiennej.

	<code>byte b;</code>	rezerwacja obszaru pamięci (deklaracja zmiennej)
	<code>b = 5;</code>	przypisanie wartości do zarezerwowanego obszaru (inicjalizacja zmiennej)

Możliwe jest również jednoczesne wykonanie obu wymienionych powyżej operacji (deklaracja i inicjalizacja zmiennej):

```
byte b = 5;
```

ZMIENNE TYPU KLASOWEGO

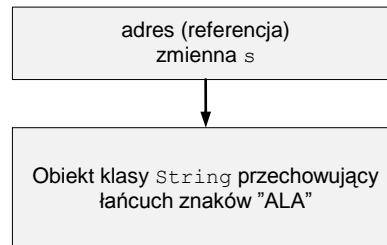
Tworząc dowolny obiekt należy zapamiętać miejsce w pamięci, w którym się on znajduje. Adres ten jest dostępny w zmiennej typu klasowego. Jest to tak zwana referencja²³ do obiektu. Deklaracja zmiennej typu obiektowego rezerwuje w pamięci miejsce przeznaczone na przechowywanie referencji do nowo tworzonego obiektu:

²³ W innych językach programowania wykorzystuje się pojęcie wskaźnika (ang. *pointer*).

```
String s;
```

natomiast użycie operatora `new` powoduje utworzenie w pamięci obiektu, którego adres zostaje przypisany do zmiennej referencyjnej

```
s = new String("ALA");
```



Możliwa jest także jednoczesna deklaracja zmiennej referencyjnej, utworzenie nowego obiektu oraz przypisanie jego adresu (położenia w pamięci) do zmiennej referencyjnej.

```
/* Jednoczesne wykonanie obu operacji (deklaracja i inicjalizacja) */
String s = new String("ALA");
```

TWORZENIE OBIEKTÓW

Obiekty w Javie tworzone są za pomocą operatora `new`, po którym występuje konstruktor²⁴ obiektu. Powstanie obiektu związane jest z przydzieleniem odpowiedniej ilości miejsca w pamięci oraz inicjalizacją pól obiektu. Po wykonaniu tych operacji zwracana jest referencja do nowo utworzonego obiektu. Poniżej przedstawiono przykładowe utworzenie obiektu klasy `Student`:

```
Student andrzej = new Student();
```

Zmienna `andrzej` zawiera referencję do miejsca w pamięci, gdzie znajduje się utworzony obiekt. Referencja ta umożliwia dostęp do pól i metod obiektu. W celu odwołania do dowolnej składowej obiektu (pola lub metody) należy określić nazwę zmiennej referencyjnej oraz nazwę dowolnej składowej obiektu (metody lub pola) oddzielonej znakiem kropki. Poniższy przykład ilustruje odwołanie do metody `pobierzWiek()`, zawartej w klasie `Student`:

```
andrzej.pobierzWiek();
```

Należy zauważyć, iż modyfikacja wartości zmiennej typu klasowego przy wykorzystaniu operatora przypisania spowoduje skopiowanie wyłącznie referencji do wskazywanego obiektu (obiekt nie jest kopiowany).

```
Student ania; // deklaracja zmiennej typu klasowego
ania = andrzej; // skopiowanie referencji (zmienna ania wskazuje na ten sam obiekt)
```

Porównanie dwóch zmiennych typu klasowego (przy użyciu operatora `==` czy `!=`) to w rzeczywistości porównanie referencji do obiektów (porównanie, czy obydwie zmienne wskazują na ten sam obiekt). Sprawdzenie czy zawartość dwóch obiektów jest identyczna odbywa się z wykorzystaniem odpowiednich metod dostępnych w danej klasie.

²⁴ Konstruktor to specjalna metoda o nazwie identycznej z nazwą klasy.

```
String napis1 = "Jan Kowalski"; // utworzenie obiektów25 i referencji
String napis2 = new String("Jan Kowalski");

if (napis1 == napis2) // porównanie referencji
    System.out.println("TAK");
else
    System.out.println("NIE");

if (napis1.equals(napis2)) // porównanie zawartości obiektów, metoda equals()
    System.out.println("TAK");
else
    System.out.println("NIE");
```

USUWANIE OBIEKTÓW

Obiekty utworzone w czasie trwania programu istnieją tak długo, jak długo istnieją do nich odwołania. Java okresowo sprawdza referencje do obiektów – te, które nie są już wykorzystywane są usuwane²⁶. Usuwanie obiektów odbywa się automatycznie. Programista nie musi dbać o zwalnianie obszaru pamięci zajmowanego przez obiekt. Zajmuje się tym specjalny proces (ang. *garbage collection*).

TWORZENIE METOD

Wydzielenie kodu programu zawierającego instrukcje pozwala na powstanie podprogramu. W Javie podprogramy nazywane są metodami²⁷. Metoda to zbiór instrukcji ujętych w nawiasy klamrowe²⁸. Każda metoda posiada modyfikator²⁹, nazwę, listę parametrów³⁰ oraz informację o typie zwracanej wartości. Nazwa metody wraz z listą parametrów (ich typami) jest często nazywana sygnaturą metody³¹ (ang. *method signature*). Każda z istniejących metod musi zostać zdefiniowana wewnątrz tworzonej klasy. Poniższy kod programu zawiera przykładowe definicje metod.

```
int pobierzWiek() {
    // int - typ zwracanej wartości do miejsca wywołania metody
    // pobierzWiek - nazwa metody
    // () - lista parametrów (pusta)
    ... // instrukcje metody
    ...
}

void ustawLiczbe(int liczba) {
    // void - metoda nie zwraca wartości
    // ustawLiczbe - nazwa metody
    // (int liczba) - lista parametrów
    ... // instrukcje metody
    ...
}

String dodajLiczbeDoLancucha(String lancuch, int liczba) {
    // String - typ zwracanej wartości (referencja do obiektu klasy String)
    // dodajLiczbeDoLancucha - nazwa metody
    // (String lancuch, int liczba) - lista parametrów
    ... // instrukcje metody
    ...
}
```

ZWRACANIE WARTOŚCI PRZEZ METODY

Każda metoda składa się ze skończonej liczby instrukcji. Działanie metody kończy się, gdy zachodzi jeden z przypadków:

- została wykonana ostatnia instrukcja metody,
- wystąpiła instrukcja `return`,

²⁵ Obiekty klasy `String` można tworzyć na dwa sposoby: przy wykorzystaniu konstruktora

`String napis = new String("Ala")` lub stosując uproszczoną dla klasy `String` formę zapisu

`String napis = "Ala";` Dodatkowo każdy łańcuch tekstowy jest traktowany przez Javę jako obiekt klasy `String`.

²⁶ Możliwe jest ręczne usunięcie referencji do obiektu poprzez przypisanie do zmiennej typu klasowego referencji pustej np. `andrzej = null`.

²⁷ W innych językach programowania podprogramy związane są z takimi pojęciami jak funkcja, czy procedura.

²⁸ Instrukcje zawarte w metodzie pomiędzy nawiasami klamrowymi nazywane są ciałem metody (ang. *body*).

²⁹ Szczegółowe omówienie modyfikatorów dostępne jest w dalszej części pracy.

³⁰ Lista parametrów może być pusta. Możliwe jest również stworzenie zmiennej liczby parametrów.

³¹ Dość często stosuje się również pojęcie nagłówka metody.

- wystąpił tzw. wyjątek³².

Metody, które zwracają wartości muszą zawierać instrukcję `return`³³, po której występuje wyrażenie określające zwracaną wartość. Typ wyrażenia musi być zgodny z typem zadeklarowanym w nagłówku metody. Instrukcja `return` zwraca wartość wyrażenia do miejsca wywołania metody.

```
int pobierzWiek() {
    int wiek;
    wiek = 23;
    return wiek; // zakończenie działania metody i zwrócenie wartości (23)
}

int obliczKwadrat(int n) {
    return (n * n); // obliczenie i zwrócenie wartości wyrażenia (n * n)
}
```

WYWOŁANIE METOD

W języku programowania Java wyróżnić można dwa rodzaje metod: związane z daną instancją³⁴ klasy oraz klasowe³⁵. Metody instancyjne nie posiadają modyfikatora `static` i wywołanie ich jest możliwe tylko na rzecz utworzonych obiektów danej klasy (bądź też z poziomu innych metod instancyjnych występujących w tej klasie). Mówiąc inaczej, aby wywołać metodę instancyjną należy posłużyć się zmienną zawierającą referencję to utworzonego wcześniej obiektu.

```
Student marek = new Student(); // utworzenie obiektu
int wiek = marek.pobierzWiek(); // wywołanie metody instancyjnej na rzecz obiektu
```

Kolejny rodzaj to metody klasowe, w których nazwa poprzedzona jest modyfikatorem `static`. Ich wywołanie jest możliwe bez konieczności tworzenia obiektu³⁶. Konieczne jest jedynie podanie nazwy klasy oraz nazwy wywoływanej metody. Poniższy przykład zawiera wywołanie metody klasowej `random()`, zawartej w klasie `java.lang.Math`.

```
int los = (int)(Math.random() * 49 + 1); // losowa wartość z przedziału <1,49>
```

PRZEKAZYWANIE DANYCH DO METOD

Wszystkie parametry w Javie przekazywane są poprzez wartość. W przypadku parametrów typu prymitywnego tworzona jest kopia wartości wewnątrz ciała metody. Instrukcje metody mogą modyfikować jedynie kopię, nie zaś wartość oryginalną, która pozostaje niezmienna. Gdy parametrem metody jest zmienna obiektowa, tworzona jest kopia referencji do obiektu, jednakże jej użycie w instrukcjach metody powoduje wykonanie operacji bezpośrednio na obiekcie (nie jest tworzona kopia obiektu wewnątrz metody).

```
void dodajLiczby(int a, int b) { // utworzenie metody
    int suma = a + b; // użycie kopii parametrów (zmienne lokalne a i b)
    a++; // modyfikacja zmiennej a
    b -= 5; // modyfikacja zmiennej b
}

// wywołanie metody z ciała innej metody
int x = 4;
int y = 2;
dodajLiczby(x, y); // wywołanie metody i przekazanie jej wartości zmiennych x i y
// wartości zmiennych x i y nie zmieniły się
System.out.printf("Wartość zmiennej x to %d, a y to %d", x, y);
```

MODYFIKATORY DOSTĘPU DO METOD

Pojęcie modyfikatora metody związane jest z określeniem zakresu jej widoczności. Poniższa tabela przedstawia zasady użycia modyfikatorów określające reguły dostępu do składowych klasy (pól i metod)

³² Szczegółowe informacje dotyczące wyjątków zostaną podane w dalszej części pracy.

³³ Instrukcja `return` nie musi być ostatnią w metodzie. Często wykorzystywana jest w instrukcjach warunkowych, decydując o wykonaniu kolejnych instrukcji metody.

³⁴ Podczas tworzenia nowych obiektów przy pomocy słowa kluczowego `new` powstaje nowa instancja (obiekt danej klasy).

³⁵ Inną nazwa metod klasowych to metody statyczne (posiadają w nagłówku modyfikator `static`).

³⁶ Metoda `main()` jest przykładem metody klasowej, która jest wywoływana bez konieczności tworzenia obiektu.

z poziomu innej klasy, pakietu (zbioru klas), klasy dziedziczącej (podklasy) oraz dowolnego miejsca programu.

Tabela 1. Modyfikatory dostępu do składowych obiektu.

MODYFIKATOR	KLASA	PAKIET	PODKLASA	WSZYSCY
public	✓	✓	✓	✓
protected	✓	✓	✓	
brak modyfikatora ³⁷	✓	✓		
private	✓			

PRZECIĄŻANIE METOD

Deklaracja wielu metod (w obrębie tej samej klasy), które posiadają identyczną nazwę, ale inny zestaw parametrów, nazywana jest przeciążaniem metod (ang. *method overloading*). Metody przeciążone rozróżniane są przez kompilator na podstawie liczby oraz typu parametrów³⁸. Takie podejście do tworzenia metod pozwala na większą elastyczność w programowaniu zorientowanym obiektowo.

```
void wyswietlNapis(String napis) {
    ... // instrukcje
}

void wyswietlNapis(String napis, int liczbaTabulacji) {
    ... // instrukcje
}

void wyswietlNapis(String napis, String extra) {
    ... // instrukcje
}
```

Powyższe metody posiadają różne sygnatury:

```
wyswietlNapis(String), wyswietlNapis(String, int), wyswietlNapis(String, String)
```

Uruchomienie wybranej metody związane jest ze sprawdzeniem liczby oraz typu jej parametrów. Przykładowo wywołanie:

```
wyswietlNapis("Informatyka i Ekonometria", 7);
```

uruchomi metodę o sygnaturze `wyswietlNapis(String, int)`.

UŻYCIE KLAS BIBLIOTECZNYCH

JDK zawiera pokaźny zbiór klas bibliotecznych. Ze względu na ich liczebność, zostały one pogrupowane w pakiety (ang. *package*). Podstawowym pakietem dostępnym w Javie jest `java.lang`³⁹. Zawiera on najbardziej przydatne i najczęściej wykorzystywane klasy (np. klasa `System`). Wykorzystanie klas znajdujących się w innych pakietach wiąże się z powiadomieniem o tym fakcie kompilatora poprzez użycie na początku kodu programu słowa kluczowego `import`⁴⁰. Konstrukcja ta występuje w dwóch wersjach, pojedynczego typu oraz typu na żądanie. W pierwszym przypadku wskazywana jest klasa, która może zostać użyta w programie.

³⁷ Brak modyfikatora oznacza zastosowanie modyfikatora domyślnego (`package`).

³⁸ Metody nie mogą być przeciążane wyłącznie poprzez określenie różnych typów zwracanych wartości.

³⁹ Wszystkie klasy zawarte w pakiecie `java.lang` są dostępne dla każdego programu bez konieczności użycia instrukcji `import`.

⁴⁰ Bez dyrektywy `import`, konieczne staje się poprzedzenie nazwy klasy nazwą pakietu, a którym się ona znajduje (np. `java.util.Scanner wejscie = new java.util.Scanner(System.in);`) Zapis taki jest wykorzystywany tylko w przypadku korzystania z klas o identycznych nazwach, ale znajdujących się w różnych pakietach.

```
import java.util.Scanner; // wykorzystanie klasy Scanner
```

Drugi rodzaj umożliwia określenie pakietu, w którym znajdują się wykorzystywane przez programistę klasy⁴¹. W czasie kompilacji właśnie w nim kompilator będzie szukał użytych klas.

```
import java.util.*; // wykorzystanie dowolnej liczby klas z pakietu java.util
```

Pierwsza forma zapisu stosowana jest w przypadku użycia jednej klasy z wybranego pakietu. Drugi ze sposobów jest preferowany w sytuacji wykorzystania większej liczby klas. Najczęściej wykorzystywane pakiety w języku programowania Java to:

- `java.lang` – podstawowe klasy wykorzystywane podczas tworzenia programów,
- `java.util` – zbiór klas pomocniczych (np. klasy do obsługi daty, czasu, kolekcji),
- `java.io` – obsługa strumieni wejścia i wyjścia, obsługa systemu plików.

Wybrane klasy biblioteczne, powszechnie używane podczas tworzenia programów:

- `System` – metody związane z właściwościami systemu, w którym uruchamiany jest program
- `Scanner` – odczyt wartości prymitywnych i łańcuchów tekstowych, często ze standardowego wejścia
- `Math` – zbiór metod realizujących podstawowe operacje arytmetyczne,
- `String` – reprezentacja łańcuchów tekstowych, zawiera wiele przydatnych metod pozwalających na operacje na ciągach znaków,
- `Arrays` – zbiór metod umożliwiających operację na tablicach (sortowanie, wyszukiwanie, itp.),
- `Calendar` – podstawowe operacje związane z datami.

KLASY OPAKOWUJĄCE

Język Java zawiera zbiór typów prostych (prymitywnych). Pomimo, iż sprawdzają się one doskonale w podstawowych operacjach, ze względu na obiektowy charakter języka Java, powstały tak zwane klasy opakowujące (ang. *wrapper classes*). Ich zadaniem jest umożliwienie przechowywania wartości typu prostego w formie obiektu. Każdy z typów prostych posiada swoją klasę opakowującą (`int` – `Integer`, `short` – `Short`, `byte` – `Byte`, `long` – `Long`, `double` – `Double`, `float` – `Float`, `boolean` – `Boolean`, `char` – `Character`). Dodatkowo każda z klas opakowujących posiada szereg metod ułatwiających operację na danych (np. metody konwersji, sprawdzania poprawności wartości). Przykładowo, klasa `Integer` przechowująca dane typu `int` posiada metodę `signum()` sprawdzającą znak liczby przekazanej jako parametr (metoda zwróci `-1` dla liczb ujemnych, `0` dla zera i `1` dla liczb dodatnich).

Pytania sprawdzające

1. Wskaż różnice, jakie występują pomiędzy zmienną typu prostego, a zmienną typu klasowego.
2. Podaj sposób tworzenia obiektów w Javie.
3. Wyjaśnij, czym jest referencja do obiektu.
4. W jaki sposób utworzone obiekty usuwane są z pamięci operacyjnej? Kiedy to następuje?
5. Podaj określenie typu dla metod, które nie zwracają żadnej wartości.
6. Wskaż różnice pomiędzy metodą instancyjną, a klasową.
7. W jaki sposób kompilator rozpoznaje metody przeciążone?
8. Jaka instrukcja kończy działanie metody?
9. Czym różnią się poszczególne modyfikatory dostępu do metod?
10. Czy poniższy kod programu to wywołanie metody instancyjnej, czy klasowej? Uzasadnij odpowiedź.

⁴¹ Kompilator wykorzysta tylko informację o klasach, które rzeczywiście zostaną użyte w kodzie programu, a nie o wszystkich klasach znajdujących się w konkretnym pakiecie.

```
double liczba = Math.sqrt(13.67);
```

11. W jaki sposób kompilator jest informowany o klasach wykorzystywanych w programie?
12. Zapoznaj się z klasami dostępnymi w pakiecie `java.util` i `java.lang`. Podaj zastosowanie klasy `StringTokenizer` oraz `StringBuilder`.
13. Dlaczego metody klasy `Math` zostały zdefiniowane jako klasowe? Uzasadnij odpowiedź.
14. Poniżej podane zostały przykładowe nagłówki metody przeciążonej `wlaczZawor()`. Wskaż ewentualne błędy w podanym zapisie.

```
boolean wlaczZawor(String nazwa, int typ)
int wlaczZawor(String identyfikator, int numer)
```

15. Sprawdź, jakie jest przeznaczenie klasy `java.util.Locale`. Podaj nazwę pakietu, w skład którego wchodzi podana klasa.
16. Jakie są różnice pomiędzy typem `double`, a `Double`? Który z wymienionych posiada większe możliwości?
17. Które z poniższych klas należą do kategorii klas opakowujących?

```
java.lang.Void, java.lang.Int, java.lang.Boolean,
java.lang.Long, java.lang.String, java.lang.Char
```

18. Jakiego typu wartości zwraca metoda `ceil(double)` z klasy `Math`?

Zadania do wykonania

Tworzenie metod

Zadanie 64 – *TworzenieMetodStatycznych.java*

Napisz program, w skład którego wchodzi dwie metody statyczne. Pierwsza z nich o sygnaturze `wyswietlTekst(String)`, wyświetlająca przekazany jako parametr łańcuch tekstowy, natomiast druga o sygnaturze `odczytajLiczbe()`, zwracająca pobraną od użytkownika liczbę typu `int`.

Rozwiązanie

```

import java.util.Scanner;

public class TworzenieMetodStatycznych {

    static void wyswietlTekst(String s) {
        System.out.print(s);
    }

    static int odczytajLiczbe() {
        Scanner klaw = new Scanner (System.in);
        return klaw.nextInt();
    }

    public static void main(String[] args) {
        int liczbaPierwsza, liczbaDruga, wynik;

        wyswietlTekst("Podaj pierwszą liczbę: "); // wywołanie metody statycznej
        liczbaPierwsza = odczytajLiczbe(); // wywołanie metody statycznej

        wyswietlTekst("Podaj drugą liczbę: "); // wywołanie metody statycznej
        liczbaDruga = odczytajLiczbe(); // wywołanie metody statycznej

        wynik = liczbaPierwsza + liczbaDruga;

        wyswietlTekst("Suma wynosi: " + wynik + "\n"); // wywołanie metody statycznej
    }
}

```

Zadanie 65 – NowaMetodaStatyczna.java

Uzupełnij kod programu zawarty w zadaniu `TworzenieMetodStatycznych.java` o kolejną metodę statyczną o sygnaturze:

```
static int odczytajLiczbe(String)
```

Dodana metoda powinna zwracać pobraną z konsoli wartość typu `int` po uprzednim wyświetleniu tekstu zawartego w parametrze metody.

Zadanie 66 – LiczbaDoskonała.java

Liczba doskonała to liczba równa sumie swoich dzielników mniejszych od niej samej. Napisz program, który sprawdza, czy podana liczba naturalna jest doskonała. Wykorzystaj metodę statyczną zwracającą wartość typu `boolean`.

Rozwiązanie

Przykład liczby doskonałej:

$$28 = 1 + 2 + 4 + 7 + 14$$
Zadanie 67 – InwersjaWektora.java

Elementy tablicy zawierają wartości typu `int`. Utwórz program, zadaniem którego będzie dokonanie inwersji (odwrócenie porządku) elementów tablicy. Napisz metodę statyczną, która wykona inwersję wartości elementów tablicy przekazanej jako parametr.

Rozwiązanie

```

public class InwersjaWektora {

    static void inwersja(int[] tablica) { // utworzenie metody statycznej
        int i = 0;
        int j = tablica.length - 1;
        int pomoc;
        while (i < j) {
            pomoc = tablica[i];
            tablica[i] = tablica[j];
            tablica[j] = pomoc;
            i++;
            j--;
        }
    }

    public static void main(String[] args) {
        int[] wektor = {4, 2, 7, 3, 9, 1, 8, 5}; // utworzenie wektora

        inwersja(wektor); // wywołanie metody statycznej

        for(int liczba : wektor) // wyświetlenie zawartości wektora
            System.out.print(liczba + " ");
    }
}

```

Zadanie 68 – WyświetlanieWektora.java

Zmodyfikuj program `InwersjaWektora.java`, dodając nową metodę statyczną `wyswietlWektor(int[])`. Korzystając z utworzonej metody, wyświetl wektor na konsoli, zarówno przed, jak i po dokonaniu inwersji.

Zadanie 69 – AnalizatorKursowWalut.java

Odszukaj w sieci Internet bieżący kurs waluty Euro dostępny w kantorach w pięciu wybranych miastach w Polsce. Napisz program, który dla podanych kursów Euro wprowadzonych z konsoli wyznaczy podstawowe miary statystyczne: minimum, maksimum, średnia arytmetyczna, mediana, średnia geometryczna oraz średnia harmoniczna.. Wykorzystaj tablicę oraz utwórz niezbędne metody statyczne.

Zadanie 70 – WeryfikacjaPeselMetoda.java

Zmodyfikuj program `WeryfikacjaPesel.java`. Uzupełnij kod programu o metodę statyczną, która dla podanego argumentu typu `String` reprezentującego numer PESEL zwróci wartość prawdy bądź fałszu w zależności od poprawności cyfry kontrolnej.

Zadanie 71 – WzorHeron.java

Pole powierzchni trójkąta S można obliczyć na podstawie długości boków a, b, c korzystając ze wzoru Herona:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad \text{gdzie } p = \frac{1}{2}(a+b+c)$$

Napisz program wyznaczający pole powierzchni trójkąta na podstawie powyższego wzoru. Utwórz metodę obliczającą pole powierzchni na podstawie przekazanych wartości reprezentujących długości boków trójkąta.

Zadanie 72 – MetodaInstancyjna.java

Napisz program wyznaczający sześcian dowolnej liczby całkowitej. Utwórz metodę instancyjną realizującą tę funkcję.

Rozwiązanie

W pierwszej kolejności powinna zostać ustalona sygnatura metody `obliczSzescian(int)` oraz typ zwracanej wartości `int`. Zwróć uwagę na brak modyfikatora `static` w nagłówku metody.

```

public class MetodaInstancyjna {

    int obliczSzescian(int liczba) { // tworzymy metode
        return liczba * liczba * liczba;
    }

    public static void main(String[] args) {
        int n = 3;
        MetodaInstancyjna obiekt = new MetodaInstancyjna (); // tworzymy obiekt
        int wynik = obiekt.obliczSzescian(n); // wywołujemy metode
        System.out.printf("Sześcian liczby %d wynosi %d", n, wynik);
    }
}

```

Zadanie 73 – *OdchylenieStandardowe.java*

Odchylenie standardowe jest jedną z najczęściej używanych miar zmienności określającą rozproszenie danych względem średniej arytmetycznej. Napisz program wyznaczający odchylenie standardowe dla dowolnej liczby argumentów rzeczywistych odczytanych z konsoli. Dla realizacji zadania utwórz metodę instancyjną o sygnaturze `odchylenieStandardowe(double[])` zwracającą wartość odchylenia standardowego dla przekazanego jako parametr wektora liczb rzeczywistych.

Rozwiązanie

Wartość odchylenia standardowego wyznacz z poniższego wzoru. M oznacza średnią arytmetyczną ze wszystkich wczytanych wartości rzeczywistych.

$$s = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2 - M^2}$$

Zadanie 74 – *MiesiacSloownie.java*

Każdy miesiąc roku kalendarzowego można wyrazić za pomocą jego nazwy lub liczby określającej pozycję miesiąca w roku. Napisz program, który na podstawie oznaczenia liczbowego miesiąca (wartości od 1 do 12) poda jego nazwę słowną. Utwórz odpowiednią metodę statyczną.

Zadanie 75 – *Totolotek.java*

Totolotek jest popularną grą liczbową polegającą na prawidłowym wytypowaniu sześciu różnych liczb naturalnych z 49 możliwych. Napisz program, który stanowił będzie symulator działania tej gry.

Rozwiązanie

Program powinien umożliwiać wprowadzenie z konsoli sześciu dowolnych liczb naturalnych, które zostaną następnie porównane z uprzednio wylosowanymi sześcioma różnymi wartościami naturalnymi z przedziału od 1 do 49. Wyświetl na konsoli liczby wprowadzone przez użytkownika, liczby wylosowane przez komputer oraz liczbę trafień (ilość liczb wprowadzonych przez użytkownika, które są zgodne z liczbami wylosowanymi przez komputer). Zrealizuj zadanie tworząc niezbędne metody statyczne.

Zadanie 76 – *KonwerterLiczbRzymskichNaArabskie.java*

Napisz podprogram (w postaci metody statycznej) `zmienRzymskaNaArabska(String)` pozwalający na zamianę liczby wyrażonej w notacji rzymskiej na arabską. Metoda powinna zwracać wartość całkowitą odpowiadającą podanej jako parametr liczbie w zapisie rzymskim lub wartość `-1`, gdy podany łańcuch znaków nie przedstawia poprawnie zapisanej liczby w notacji rzymskiej. Utworzony program do konwersji liczb powinien dokonywać poprawnego przekształcenia dla każdej wartości z przedziału od I do MMMCMXCIX.

Rozwiązanie

Zapoznaj się z opisem rzymskiego systemu liczbowego, dostępnego pod adresem: http://pl.wikipedia.org/wiki/Rzymski_system_liczbowy

Zadanie 77 – *Statki.java*

„Okrety” („Statki”) należą do gatunku gier strategicznych polegających na zatopieniu floty przeciwnika bez znajomości dokładnego jej położenia. Ze względu na swą prostotę, może ona być prowadzona z użyciem kartki i ołówka. Napisz program realizujący funkcje gry przy użyciu komputera.

Rozwiązanie

Zapoznaj się z zasadami gry, które znajdziesz w sieci Internet. Utwórz tablicę dwuwymiarową 10x10, w której będziesz przechowywał pozycje poszczególnych okrętów. Napisz metodę losowo ustalającą początkową pozycję statków. Przyjmij, iż flota składa się z sześciu różnych statków. Gra polega na odczytywaniu z konsoli współrzędnych określających położenie okrętu oraz wyświetlaniu informacji, czy statek został trafiony (komunikaty: pudło, trafiony, zatopiony). Program kończy działanie, gdy zatopione zostały wszystkie okręty, wyświetlając liczbę podjętych prób trafień oraz czas (w minutach i sekundach), w którym gra została zrealizowana.

WYKORZYSTANIE KLAS BIBLIOTECZNYCH**Zadanie 78 – DataOrazCzas.java**

Zapoznaj się z działaniem klasy `java.util.Calendar`. Napisz program, który wyświetli: aktualną datę i godzinę, liczbę dni oraz tygodni, jakie upłynęły od początku roku, liczbę dni oraz tygodni, jaka pozostała do końca roku, liczbę lat, miesięcy, dni, godzin, minut i sekund, jaką przeżyłeś dotychczas.

Zadanie 79 – KalendarzMiesieczny.java

Napisz program wyświetlający kalendarz miesięczny dla podanego jako parametr roku oraz miesiąca. Utwórz metodę statyczną `wyswietlKalendarz(int, int)` realizującą funkcję wyświetlania kalendarza na konsoli (przykładowe wywołanie metody: `wyswietlKalendarz(2009, 3)`).

```
Marzec 2009
Pn Wt Śr Cz Pt Sb Nd
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Zadanie 80 – WartoscSprzedazy.java

Firma MIKOM sp. z o.o. produkuje odkurzacze domowe. Ich sprzedażą zajmuje się dziesięciu pracowników – przedstawicieli handlowych firmy. Napisz program, który umożliwi pobranie z konsoli informacji o łącznej wartości sprzedaży uzyskanej przez każdego z pracowników. Wyświetl na konsoli wprowadzone wartości w porządku malejącym. Utwórz własne metody oraz wykorzystaj metody dostępne w klasie `java.util.Arrays`.

Zadanie 81 – ZaokraglanieWartosci.java

Poniższy program przedstawia wykorzystanie kilku metod służących do zaokrąglania wartości rzeczywistych. Które linijki kodu programu wyświetlą dokładnie wartość 11? Podaj uzasadnienie.

```
public class ZaokraglanieWartosci {
    public static void main(String[] args) {
        double v = 10.5;
        System.out.println(Math.ceil(v));
        System.out.println(Math.round(v));
        System.out.println(Math.floor(v));
        System.out.println((int) Math.ceil(v));
        System.out.println((int) Math.floor(v));
    }
}
```

Zadanie 82 – FunkcjeTrygonometryczne.java

Trygonometria stanowi dział matematyki zajmujący się badaniem związków między bokami i kątami w trójkącie. Korzystając z metod zawartych w klasie `Math` utwórz własne metody `cosinus(double)`, `sinus(double)`, `tangens(double)` realizujące wymienione funkcje trygonometryczne. Parametrami metod powinny być wartości miar kąta wyrażone w stopniach. Napisz program wykorzystujący utworzone metody.

Zadanie 83 – KonkatencjaLancuchow.java

Poniższy program przedstawia wykorzystanie metody `concat()` służącej do złączania łańcuchów znakowych. Jaki będzie rezultat uruchomienia programu?

```

public class KonkatenacjaLancuchow {
    public static void main(String[] args) {
        String napis1 = "Uniwersytet";
        String napis2 = "Ekonomiczny";
        String napis3 = "w Krakowie";
        napis1.concat(napis2);
        napis1.concat(napis3.concat(napis1));
        System.out.println(napis1);
    }
}

```

Zadanie 84 – SystemyLiczbowe.java

Zapoznaj się z metodą `valueOf(String, int)` występującą między innymi w klasie opakowującej `Integer`. Napisz program odczytujący z konsoli liczbę w postaci łańcucha tekstowego oraz wielkość podstawy systemu liczbowego. Wynikiem działania programu powinna być liczba w systemie dziesiętnym.

Zadanie 85 – ObjetoscStozka.java

Objętość stożka wyraża się wzorem:

$$V = \frac{1}{3} \pi r^2 h$$

Napisz program wyznaczający objętość stożka. Wartości promienia podstawy `r` oraz wysokości `h` odczytaj z wiersza poleceń. Wykorzystaj stałą `PI` dostępną w klasie `Math`.

Zadanie 86 – Palindrom.java

Korzystając z klasy `java.lang.StringBuilder` utwórz program odczytujący z konsoli łańcuch tekstowy i wyświetlający go wraz z jego lustrzanym odbiciem.

Rozwiązanie

```

import java.util.*;

public class Palindrom {
    public static void main(String[] args) {
        Scanner klawiatura = new Scanner(System.in);
        System.out.print("Podaj napis: ");
        String napis1 = klawiatura.nextLine();
        StringBuilder napis2 = new StringBuilder(napis1);
        System.out.println(napis1 + napis2.reverse());
    }
}

```

Zadanie 87 – EliminatorZnakow.java

Analizie poddawane są dwa ciągi znaków. Napisz program, który z pierwszego ciągu usunie wszystkie wystąpienia znaków znajdujących się w drugim ciągu. Procedura usuwania powinna być przeprowadzona bez rozróżniania wielkości znaków.

Rozwiązanie

Poniższy przykład ilustruje działanie programu.

```

Pierwszy ciąg znaków: JANEK
Drugi ciąg znaków: aeioy
Ciąg wynikowy: JNK

```

Zadanie 88 – WyszukiwanieWymiana.java

Jedną z podstawowych funkcji każdego edytora tekstu jest możliwość wyszukania dowolnego ciągu znaków i jego wymiana na inny. Napisz program realizujący powyższą funkcję edytora.

Rozwiązanie

Program odczytuje z konsoli 3 ciągi znaków, a następnie modyfikuje pierwszy z nich w następujący sposób: wszystkie wystąpienia drugiego ciągu są zastępowane trzecim ciągiem. Wykorzystaj klasę `java.util.StringBuilder` oraz metody dostępne w klasie `String`. Przykładowe działanie programu przedstawiono poniżej.


```
Pierwszy ciąg: kot pies kot pies kot pies
Drugi ciąg: kot
Trzeci ciąg: kotek
Wynik działania: kotek pies kotek pies kotek pies
```

Zadanie 89 – PodziałWyrazenia.java

Napisz program dzielący na składowe dowolne wyrażenie arytmetyczne złożone z argumentów całkowitych, operatorów dodawania, odejmowania, mnożenia i dzielenia oraz nawiasów okrągłych. Wyświetl składowe wyrażenia w oddzielnych liniach.

Rozwiązanie

Poniższy przykład ilustruje działanie programu.

```
Wyrażenie: 45*(12-9)
```

```
45
*
(
12
-
9
)
```

Wskazówka

Przed podziałem wyrażenia usuń z niego wszystkie spacje. Wykorzystaj klasę `java.util.StringTokenizer` oraz metody w niej zawarte.

Rozdział 5

Klasy i obiekty

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- wskazać różnice pomiędzy klasą, a obiektem,
- prawidłowo tworzyć klasy z wykorzystaniem pól i metod,
- tworzyć i wykorzystywać konstruktory,
- efektywnie posługiwać się operatorem `this`,
- wykorzystać enkapsulację do ochrony danych.

Wprowadzenie do zagadnień

Java jest językiem, w którym realizowana jest idea programowania zorientowanego obiektowo (ang. *Object Oriented Programming*). Tworzenie aplikacji odbywa się z wykorzystaniem odpowiednich, współpracujących ze sobą elementów, zwanych obiektami, konstruowanych na podstawie wzorca zwanego klasą. Biblioteki Javy zawierają pokaźny zbiór gotowych do użycia klas, które ułatwiają tworzenie programów, jednakże złożone aplikacje składają się najczęściej z szeregu nowych klas, stworzonych przez programistę.

KLASY

W otaczającej rzeczywistości występuje szereg obiektów tego samego typu (np. zbiór samochodów, drzew, czy studentów). Klasa określana jest jako zbiór stanów oraz zachowań opisujących obiekty należące do tej samej kategorii. Jest to pewnego rodzaju wzorzec, na podstawie którego tworzone są unikalne wersje obiektu (np. czerwony samochód marki Fiat, wysokie drzewo). W skład każdej klasy wchodzi:

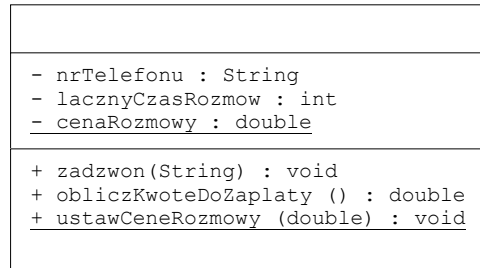
- pola (zmienne), zawierające informacje opisujące właściwości (stany) obiektów,
- metody (podprogramy), modelujące zachowania obiektów.

OBIEKTY

Na podstawie zdefiniowanej klasy tworzone są obiekty różniące się właściwościami. Każdy obiekt posiada własny zbiór pól, których wartości określają jego indywidualność. Na podstawie klasy możliwe jest zatem utworzenie dowolnej liczby obiektów (instancji klasy), należących do tej samej kategorii, zróżnicowanych pod względem właściwości.

Tworzenie klasy

Proces definiowania klasy polega na określeniu jej składowych, tj. pól oraz metod, reprezentujących stany i zachowania obiektów, które będą tworzone w oparciu o tę klasę. Poniższy przykład ilustruje klasę *Telefon* stanowiącą wzorzec dla zbioru opisywanych telefonów. Każdy telefon posiada pewne unikalne cechy (numer telefonu, łączny czas rozmów) oraz zachowania, tj. czynności, które może wykonać (zadzwoń).



Rys. 2. Diagram UML⁴² klasy Telefon.

Definicja klasy w języku Java realizowana jest za pomocą słowa kluczowego `class`:

```

class Telefon {

    // ciało klasy (składowe klasy)

}
  
```

Każda tworzona klasa może zawierać zbiór pól oraz metod stanowiących składowe klasy.

POLA KLASY

Pola to zmienne deklarowane wewnątrz klasy. W przypadku braku jawnej inicjalizacji zmiennej otrzymuje ona wartość domyślną⁴³. Zwyczajowo deklaracja pól klasy występuje przed deklaracją metod. Poniższy kod programu zawiera definicję klasy Telefon wraz z wchodzącymi w jej skład polami.

```

class Telefon {

    // deklaracja pól klasy

    private String numerTelefonu;
    private int lacznyCzasRozmow;
    private static double cenaRozmowy = 0.48; // zł/min.

}
  
```

Pola statyczne (oznaczone w kodzie modyfikatorem `static`), w przeciwieństwie do pól niestatycznych, są wspólne dla wszystkich obiektów danej klasy⁴⁴. Dostęp do nich jest możliwy bez konieczności tworzenia obiektu klasy. W powyższym przykładzie pole `cenaRozmowy` dotyczy właściwości odnoszącej się do wszystkich obiektów klasy Telefon. Dostęp do pól klasy jest możliwy zgodnie z rodzajem użytego w deklaracji modyfikatora dostępu.

METODY KLASY

Metody deklarowane w klasie stanowią odpowiednik zachowań konkretnych obiektów. Sposób deklaracji metod został omówiony w poprzednim module. Warto przypomnieć o metodach statycznych (klasowych), do których dostęp możliwy jest bez konieczności tworzenia obiektu danej klasy. Metody te nie mogą zatem odwoływać się do niestatycznych (instancyjnych) składowych klasy (pól i metod) powiązanych z konkretnym obiektem. Poniższy kod programu zawiera definicję klasy Telefon wraz z jej składowymi – polami oraz metodami.

⁴² Szczegółowe informacje o diagramach UML dostępne są na stronie <http://www.uml.com.pl/>

⁴³ W przypadku braku jawnej inicjalizacji pól klasy, przyjmują one wartości domyślne (pola numeryczne wartości zerowe, pola logiczne wartość `false`, natomiast pola klasowe wartość `null`).

⁴⁴ Pola statyczne nazywane są dość często polami klasowymi.

```

class Telefon {

    // deklaracja pól

    private String numerTelefonu;
    private int lacznyCzasRozmow;
    private static double cenaRozmowy = 0.48;    // zł/min.

    // deklaracja metod

    public double obliczKwoteDoZaplaty() {
        return cenaRozmowy * (lacznyCzasRozmow / 60);
    }

    public static void ustawCeneRozmowy(double nowaCena) {
        cenaRozmowy = nowaCena;
    }

    public void zadzwon(String nrTelefonu) {
        System.out.println ("Dzwonię do: " + nrTelefonu);
        System.out.println ("Dryń, dryń...");
        System.out.println ("Rozmowa w toku...");
        int czasRozmowy = (int) (Math.random()*3600);
        lacznyCzasRozmow += czasRozmowy;
        System.out.println ("Rozmowa zakończona. ");
        System.out.printf ("Czas rozmowy: %d min. %d sek.",
            czasRozmowy/60, czasRozmowy%60);
    }
}

```

KONSTRUKTOR

Konstruktor stanowi specyficzną metodę, która wywoływana zostaje w momencie tworzenia obiektu⁴⁵. Jego nazwa musi być identyczna z nazwą klasy. Konstruktor nie może również zwracać żadnej wartości⁴⁶. W skład konstruktora wchodzi instrukcje, które zostaną wykonane w trakcie tworzenia obiektu. Szczególnym przypadkiem jest tu nadanie wartości początkowej polom obiektu, co może zostać zrealizowane albo w momencie deklaracji pola, albo też poprzez przypisanie wartości w konstruktorze obiektu.

Poniższy kod programu zawiera deklarację konstruktora `Telefon(String)` dla klasy `Telefon`. Jego składową jest instrukcja przypisująca wartość początkową dla pola `numerTelefonu`.

```

public Telefon (String numer) {
    numerTelefonu = numer;
}

```

W przypadku braku jawnie zadeklarowanego konstruktora Java automatycznie wywołuje konstruktor domyślny⁴⁷. Możliwe jest przeciążanie konstruktorów, podobnie jak innych metod zadeklarowanych w klasie.

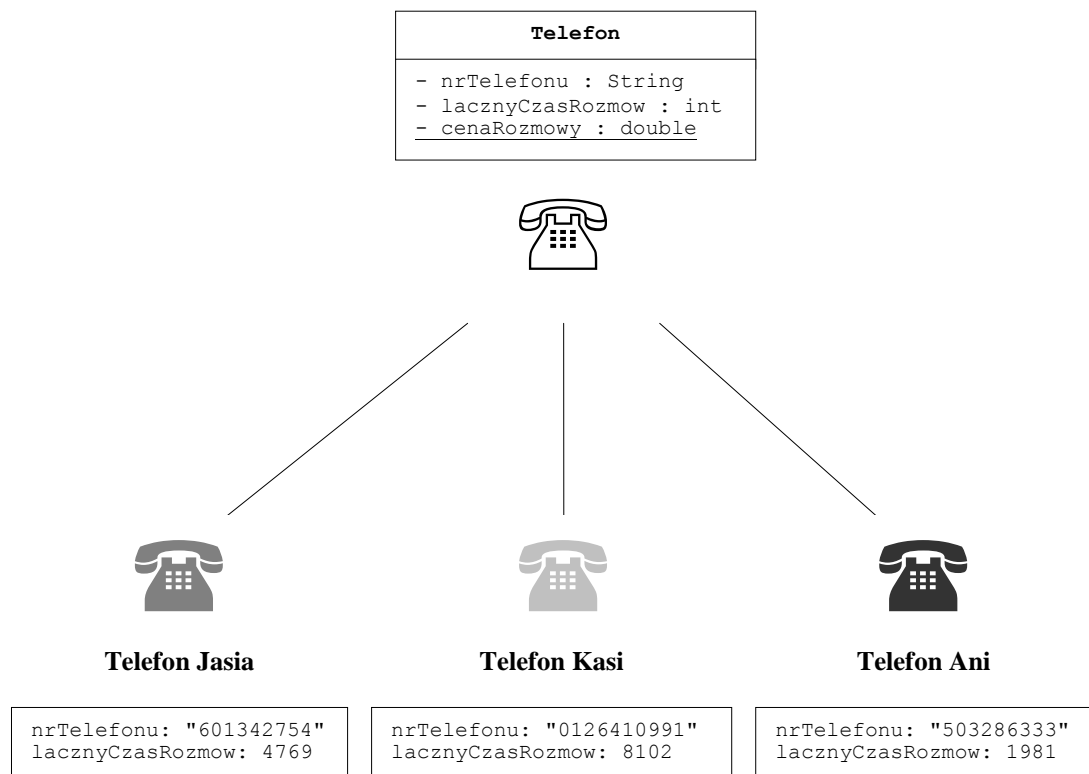
Tworzenie obiektów

Definicja klasy udostępnia wzorzec, na podstawie którego tworzone są obiekty występujące w programie. Ich interakcja realizowana poprzez wywoływanie metod stanowi o treści tworzonej aplikacji. Poniższy rysunek przedstawia schemat tworzenia obiektów na podstawie klasy `Telefon`.

⁴⁵ Składnia tworzenia obiektu zawiera operator `new` oraz nazwę konstruktora, który ma zostać wywołany.

⁴⁶ Przed nazwą konstruktora nie może pojawić się również słowo kluczowe `void`.

⁴⁷ Konstruktor domyślny posiada pustą liczbę parametrów i jest wywoływany wyłącznie wtedy, gdy klasa nie posiada żadnego jawnie zadeklarowanego konstruktora.



Rys. 3. Tworzenie obiektów na podstawie klasy.

Tworzenie nowych obiektów w Javie polega na użyciu operatora `new`, po którym występuje nazwa odpowiedniego konstruktora. Poniższy kod zawiera przykład utworzenia obiektu klasy `Telefon`:

```
Telefon telefonKasi = new Telefon("0126410991");
```

Operator `new` tworzy na podstawie wzorca (klasy) nowy obiekt oraz umieszcza go w pamięci operacyjnej. Wartością zwracaną podczas tworzenia obiektu jest referencja do miejsca, w którym obiekt został utworzony. Wartość ta przypisana zostaje do zmiennej typu obiektowego, co pozwala na odwoływanie się do składowych obiektu:

```
public class WykorzystanieObiektow {
    public static void main(String args[]) {
        Telefon telefonJasia = new Telefon("601342754");
        // wywołanie metody zadzwon() na rzecz obiektu telefonJasia
        telefonJasia.zadzwon("687934267");
    }
}
```

Pojęcia dodatkowe

Definiowanie klas oraz tworzenie obiektów powiązane jest ściśle z typem referencyjnym, przechowującym odwołania do obiektów. Wykorzystując referencję oraz operator kropki uzyskujemy dostęp do składowych klasy. Kto i na jakich zasadach posiada dostęp do poszczególnych składowych określają modyfikatory dostępu. Należy zauważyć, iż dostęp do poszczególnych pól klasy odbywa się zazwyczaj za pośrednictwem zadeklarowanych metod.

METODY DOSTĘPWE I MODYFIKUJĄCE

Metody dostępne (ang. *accessor (getter) methods*) są używane do odczytu wartości pól (instancyjnych i statycznych). Nazwa takiej metody zazwyczaj składa się z czasownika „pobierz” (ang. *get*) oraz nazwy pola.

Zadaniem metod modyfikujących (ang. *mutator (setter) methods*) jest nadanie bądź też zmiana wartości pól (instancyjnych i statycznych). Nazwa metody zawiera zwykle czasownik „ustaw” (ang. *set*) oraz nazwę pola. Metody modyfikujące z reguły nie zwracają żadnej wartości.

```
class Zeszyt{
    private int liczbaKartek;

    // metoda dostępowa
    public int pobierzLiczbeKartek() {
        return liczbaKartek;
    }

    // metoda modyfikująca
    public void ustawLiczbeKartek(int liczba) {
        liczbaKartek = liczba;
    }
}
```

SŁOWO KLUCZOWE THIS

Słowo `this` oznacza referencję do bieżącego obiektu. Ułatwia to dostęp do jego składowych, jak również umożliwia wywołanie odpowiednich konstruktorów. Często `this` jest wykorzystywane, gdy nazwa parametru metody jest identyczna z nazwą pola⁴⁸, umożliwiając rozróżnienie tych identyfikatorów.

```
class Student {
    private String nazwisko;

    public void ustawNazwisko(String nazwisko) {
        // inicjalizacja pola "nazwisko" wartością parametru "nazwisko"
        this.nazwisko = nazwisko;
    }
}
```

Wywołanie konstruktora z ciała innego konstruktora jest możliwe poprzez użycie słowa kluczowego `this` wraz z ewentualną listą parametrów⁴⁹.

```
class Monitor {

    private String nazwa;
    private static int liczbaMonitorów;

    public Monitor(){
        liczbaMonitorów++;
    }

    public Monitor(String nazwa) {
        this(); // wywołanie konstruktora bezparametrowego
        this.nazwa = nazwa; // inicjalizacja pola wartością parametru
    }
}
```

OKREŚLANIE WIDOCZNOŚCI SKŁADOWYCH KLASY

Prawidłowe określenie widoczności poszczególnych składowych klasy umożliwia ukrycie implementacji. Zgodnie z zasadami programowania zorientowanego obiektowo dostęp do pól powinien być realizowany wyłącznie za pomocą metod dostępowych i modyfikujących (tworzących tzw. interfejs obiektu). Proces ukrywania składowych klasy nazywany jest enkapsulacją⁵⁰ i odbywa się poprzez użycie odpowiedniego modyfikatora dostępu (zazwyczaj `private`).

⁴⁸ Występuje wtedy tzw. przesłanianie zmiennych.

⁴⁹ Wywołanie konstruktora przy użyciu `this` musi wystąpić na początku bloku instrukcji..

⁵⁰ Enkapsulacja często nazywana jest również hermetyzacją danych.

```

class Osoba{

    // pola prywatne - dostęp możliwy tylko za pośrednictwem odpowiednich metod
    private String imie;
    private String nazwisko;
    private double pensja;

    // konstruktor
    public Osoba(String imie, String nazwisko, double pensja) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.pensja = pensja;
    }

    // publiczna metoda umożliwiająca zmianę pensji danej osoby
    public void ustawPensje(double nowaPensja) {
        pensja = nowaPensja;
    }
}

```

Pytania sprawdzające

1. Czym jest klasa oraz jakie posiada właściwości?
2. Do czego wykorzystywane są diagramy UML?
3. Czym jest obiekt oraz w jaki sposób jest tworzony?
4. Jaka rolę w programowaniu obiektowym spełnia operator `new`?
5. Wskaż różnice pomiędzy metodą, a konstruktorem.
6. Jaki proces zachodzi podczas wykonywania poniższej instrukcji?

```
Laptop laptopFirmowy = new Laptop("Lenovo");
```

7. Wymień i scharakteryzuj składowe klasy.
8. Czy niezainicjalizowane pola posiadają wartość początkową? Jeśli tak, to jaką?
9. Czy każda klasa musi posiadać konstruktor?
10. Poniższy kod programu zawiera definicję klasy wraz z jej składowymi. Wskaż ewentualne błędy.

```

class PenDrive {
    private String nazwa;
    private int pojemnosc;

    public PenDrive(String nazwa, int pojemnosc){
        nazwa = nazwa;
        pojemnosc = pojemnosc;
        return true;
    }
}

```

11. Czy możliwe jest utworzenie nowego obiektu klasy `PenDrive` w następujący sposób?

```
PenDrive mojPendrive = new PenDrive();
```

12. Jaką funkcję pełni słowo kluczowe `this`?
13. Jakie jest główne zastosowanie enkapsulacji?
14. Poniżej podany został kod programu wykorzystujący klasę `PenDrive`. Czy odwołanie do pola `pojemnosc` jest poprawne?

```

PenDrive mojPendrive = new PenDrive("Toshiba", 4096);
mojPendrive.pojemnosc = 8192;

```

15. Wskaż różnice pomiędzy statycznymi (klasowymi) i niestatycznymi (instancyjnymi) polami klasy.

16. Dlaczego z metody statycznej nie jest możliwe odwołanie do składowych instancyjnych?
17. Podaj sytuacje, w których wykorzystywane są metody prywatne.
18. Wskaż przykład zastosowania metod dostępowych oraz modyfikujących dla klasy PenDrive.
19. Jaka informacja zostanie wyświetlona na konsoli po wykonaniu poniższego kodu programu?

```
Student[] studenci = new Student[10];
System.out.println(studenci[4]);
```

20. Wskaż błąd w poniższym kodzie programu, wykorzystującym klasę Telefon.

```
int n = 10;
Telefon[] telefony = new Telefon[n];
for (int i = 0; i < n; i++) {
    telefony[i].zadzwon("112");
}
```

Zadania do wykonania

Zadanie 90 – *TelefonyKowalskich.java*

Rodzina Kowalskich to Ania i Jarek. Oboje posiadają telefony i korzystają z nich często. Przygotuj w pełni funkcjonalną klasę `Telefon` oraz sprawdź jej działanie tworząc obiekty telefonów Ani i Jarka. Wykonaj rozmowę z telefonu Ani oraz Jarka, a następnie oblicz kwotę do zapłaty, jaką powinna uiścić Ania.

Rozwiązanie

```
class Telefon {

    // deklaracja pól

    private String numerTelefonu;
    private int lacznyCzasRozmow;
    private static double cenaRozmowy = 0.48;    // zł/min.

    // konstruktor

    public Telefon (String numer) {
        numerTelefonu = numer;
    }

    // deklaracja metod

    public double obliczKwoteDoZaplaty() {
        return cenaRozmowy * (lacznyCzasRozmow / 60 + 1);
    }

    public static void ustawCeneRozmowy(double nowaCena){
        cenaRozmowy = nowaCena;
    }

    public void zadzwon(String nrTelefonu) {
        System.out.println ("Dzwonię do: " + nrTelefonu);
        System.out.println ("Dryń, dryń...");
        System.out.println ("Rozmowa w toku...");
        int czasRozmowy = (int) (Math.random()*3600);
        lacznyCzasRozmow += czasRozmowy;
        System.out.println ("Rozmowa zakończona. ");
        System.out.printf ("Czas rozmowy: %d min. %d sek. \n",
            czasRozmowy/60, czasRozmowy%60);
    }

}

public class TelefonyKowalskich {

    public static void main(String[] args){
        Telefon telefonAni = new Telefon("783982331");
        Telefon telefonJarka = new Telefon("608234982");
    }
}
```



```

        telefonAni.zadzwon("0124239832");
        telefonJarka.zadzwon("112");

        double kwota = telefonAni.obliczKwoteDoZapłaty();
        System.out.printf("Ania ma do zapłaty %f zł.", kwota);
    }
}

```

Zadanie 91 – TelefonInfo.java

Uzupełnij klasę `Telefon` o metody dostępowe oraz modyfikujące dla wszystkich pól. Sprawdź działanie utworzonych metod (np. wyświetlenie numeru telefonu, wyświetlenie łącznego czasu rozmów, zmiana numeru telefonu dla konkretnych obiektów).

Zadanie 92 – WybieraneNumery.java

Rozszerz funkcjonalność klasy `Telefon` poprzez dodanie listy dziesięciu ostatnio wybieranych numerów. Zmodyfikuj metodę `zadzwon(String)` oraz dodaj metodę wyświetlającą listę połączeń `pokazWybieraneNumery()`.

Wskazówka

Listę ostatnio wybieranych numerów możesz przechowywać w tablicy.

```
private String[] wybieraneNumery;
```

Modyfikacja tablicy powinna następować przy każdym wywołaniu metody `zadzwon(String)`

Zadanie 93 – Licznik.java

W zaawansowanych programach często istnieje konieczność dokonania zliczania liczby wykonywanych czynności. Zdefiniuj klasę `Licznik` z jednym polem prywatnym `ilosc`. Dopisz metodę dostępową zwracającą wartość pola `ilosc` oraz modyfikującą, której zadaniem będzie inkrementacja pola `ilosc`. Następnie napisz program, który wykorzysta utworzoną klasę do zliczania liczby znaków spacji w podanym przez użytkownika zdaniu.

Zadanie 94 – Stoper.java

Sprawdzanie wydajności aplikacji wiąże się z pomiarem czasu jej wykonania. Zaprojektuj klasę `Stoper`. Jej głównym zadaniem będzie pomiar czasu pomiędzy uruchomieniem stopera, a jego zatrzymaniem. Zastanów się, jakie pola i metody będą potrzebne do realizacji tego zadania. Wykorzystaj metodę `currentTimeMillis()` z klasy `System`.

Rozwiązanie

```

class Stoper {

    // deklaracja pól
    private long start;
    private long stop;
    private String nazwa;

    // konstruktory
    public Stoper() {
        this("");
    }

    public Stoper(String nazwa) {
        this.nazwa = nazwa;
    }

    // metody
    public void start(){
        start = System.currentTimeMillis();
    }

    public void stop(){
        stop = System.currentTimeMillis();
    }

    public double pobierzWynik(){
        return (stop - start) / 1000.0;
    }
}

```

Zadanie 95 – TesterWydajnosciAplikacji.java

Wykorzystaj klasę `Stoper` do sprawdzenia, jaki jest czas wyznaczania kolejnych liczb pierwszych. Napisz program, który wyznaczy 10000 liczb pierwszych poprzez sprawdzenie kolejnych liczb naturalnych N , czy podzielne są wyłącznie przez 1 i N . Następnie porównaj szybkość działania tej metody z algorytmem „sita Eratostenesa”. Który z algorytmów jest szybszy? Zastanów się nad optymalizacją kodu w celu zwiększenia wydajności.

Zadanie 96 – PomiarCzasowPosrednich.java

W większości zadań z zastosowaniem stopera niezbędny jest pomiar czasów pośrednich (tzw. międzyczas). Dodaj do klasy `Stoper` odpowiednią metodę zwracającą międzyczas osiągnięty w chwili jej wywołania. Zmodyfikuj program sprawdzający wydajność metod generujących liczby pierwsze, dokonując pomiaru i wyświetlania czasu obliczeń dla każdego z 2500 wyznaczonych liczb pierwszych.

Zadanie 97 – Kartografia.java

Współrzędne geograficzne wyrażane są za pomocą długości i szerokości geograficznej, mierzonych w stopniach, minutach i sekundach. Napisz program, który umożliwia przechowywanie informacji dotyczących położenia dowolnie wybranych miejscowości oraz pozwala na wyznaczenie odległości, które je dzielą. Wybierz 3 dowolne miejscowości (w tym miejscowość, w której mieszkasz), a następnie odszukaj w sieci Internet ich współrzędne geograficzne. Wyznacz odległości dzielące te miejscowości. Wyświetl rezultaty na konsoli.

Rozwiązanie

Utwórz klasę `WspolrzedneGeograficzne`, która stanowić będzie podstawę do tworzenia obiektów reprezentujących dowolne miejscowości. Konstruktor klasy powinien zawierać parametry określające położenie (współrzędne geograficzne) miejscowości. Utwórz metodę statyczną umożliwiającą wyznaczenie (w stopniach, minutach i sekundach) odległości dzielącej dwie dowolnie wybrane miejscowości.

Zadanie 98 – UlamkiZwykle.java

Java nie posiada klasy dla obsługi ułamków zwykłych. Zdefiniuj klasę `Ulamki` zawierającą pola licznik oraz mianownik, a także przykładowe metody (tworzenie ułamka, jego upraszczanie oraz wyświetlanie). Sprawdź działanie klasy tworząc ułamki zwykłe oraz wyświetlając je na konsoli.

Rozwiązanie

```

class Ulamek {

    // deklaracja pól

    private int licznik;
    private int mianownik;

    // konstruktory

    public Ulamek() {
        this.licznik = 0;
        this.mianownik = 1;
    }

    public Ulamek(int licznik) {
        this.licznik = licznik;
        this.mianownik = 1;
    }

    public Ulamek(int licznik, int mianownik) {
        this.licznik = licznik;
        this.mianownik = mianownik;

        uproscUlamek(this);
    }

    // metody

    private static int obliczNWD(Ulamek a) {

        int l,m;
        l = a.licznik;
        m = a.mianownik;

        if (l < 0)
            l = - l;
        if (m < 0)
            m = - m;

        while ( l != m) {
            if (l > m) l = l - m;
            else m = m - l;
        }

        return l;
    }

    private static void uproscUlamek(Ulamek a) {
        int nwd;

        nwd = obliczNWD(a);

        a.licznik /= nwd;
        a.mianownik /= nwd;
    }

    public String toString() {
        return this.licznik + "/" + this.mianownik;
    }
}

public class UlamkiZwykle {
    public static void main (String[] args) {
        Ulamek x = new Ulamek(4, 6);
        Ulamek y = new Ulamek(10, 12);

        System.out.println ("Ulamek x = " + x);
        System.out.println ("Ulamek y = " + y);
    }
}

```

Zadanie 99 – OperacjeNaUlamkach.java

Uzupełnij klasę `Ulamek` o brakujące metody umożliwiające przeprowadzenie podstawowych operacji arytmetycznych. Sygnatury metod znajdziesz poniżej. Sprawdź poprawność działania wszystkich utworzonych metod.

```
public static Ulamek iloczynUlamkow(Ulamek a, Ulamek b)
public static Ulamek sumaUlamkow(Ulamek a, Ulamek b)
public static Ulamek ilorazUlamkow(Ulamek a, Ulamek b)
public static Ulamek roznicaUlamkow(Ulamek a, Ulamek b)
public static Ulamek naZwyczajy(double a)
public static double naDziesietny(Ulamek a)
```

Zadanie 100 – OperacjeNaDataach.java

Sprawdź, ile dni już przeżyłeś, którego dnia tygodnia się urodziłeś oraz ile dni pozostało do końca roku kalendarzowego. Utwórz klasę reprezentującą datę. Użytkownik powinien posiadać możliwość utworzenia obiektów tej klasy podając dzień, miesiąc oraz rok. W skład klasy powinna wchodzić metoda obliczająca liczbę dni pomiędzy dwoma datami, zwracająca dzień tygodnia oraz podająca znak zodiaku dla podanej daty.

Zadanie 101 – LiczbyZespolone.java

Liczby zespolone to liczby w postaci $x+iy$, gdzie x i y to liczby rzeczywiste, a i jest jednostką urojoną spełniającą warunek $i^2=-1$. Liczba x jest częścią rzeczywistą liczby zespolonej, a y jej częścią urojoną. Utwórz klasę `LiczbyZespolone` umożliwiającą tworzenie liczb oraz podstawowe operacje z ich udziałem (dodawanie, odejmowanie, mnożenie, obliczanie wartości bezwzględnej, zwracanie części rzeczywistej, urojonej, wyświetlanie liczby zespolonej).

Wskazówka

Więcej informacji o liczbach zespolonych znajdziesz na stronie:

http://pl.wikipedia.org/wiki/Liczby_zespolone

Zadanie 102 – StypendiumNaukowe.java

Do uzyskania stypendium naukowego wymagana jest średnia uzyskanych ocen, większa bądź równa 4,8. Pobierz ze standardowego wejścia oceny studenta i sprawdź czy uzyskał on wymagana średnią. Skorzystaj z własnej klasy `Statystyka` obliczającej średnią arytmetyczną.

Rozwiązanie

```
import java.util.*;

class Statystyka {
    // deklaracja pól
    private double dane[];

    // konstruktory
    public Statystyka(double[] tablica) {
        dane = tablica;
    }

    // metody
    public double sredniaArytmetyczna(){
        double suma = 0;
        for (double x : dane)
            suma += x;
        return (suma / dane.length);
    }
}

public class StypendiumNaukowe{

    public static void main(String[] args){
        Scanner wejscie = new Scanner(System.in);
        System.out.print("Podaj ilość ocen: ");
        int n = wejscie.nextInt();
        double[] oceny = new double[n];
        for(int i=0; i<n; i++){
            System.out.print("Podaj ocenę: ");
            oceny[i] = wejscie.nextDouble();
        }
        Statystyka obliczenia = new Statystyka(oceny);
```

```

        double srednia = obliczenia.sredniaArytmetyczna();
        System.out.println("Stypendium naukowe " + (srednia >= 4.8 ?
            "zostało przyznane" : "nie zostało przyznane"));
    }
}

```

Zadanie 103 – ObliczeniaStatystyczne.java

Szef Ani poprosił ją o wyznaczenie podstawowych wskaźników statystycznych dla kilku ciągów danych. Poniżej zamieszczone zostały dane dostarczone przez szefa Ani. Dodaj do klasy Statystyka niezbędne metody (mediana, dominanta, odchylenie średnie, wariancja, odchylenie standardowe) oraz przedstaw uzyskane rezultaty.

```

Klient 1: 0.5, 4.7, 3.8, 7.9, 5.2, 6.0, 3.5, 7.1
Klient 2: 9.3, 2.6, 6.3, 5.2, 3.5, 8.3, 8.1, 3.0, 3.1, 5.2, 7.7
Klient 3: 2.1, 2.5, 2.8, 3.4, 4.5, 3.9, 5.6, 6.0, 5.7, 3.4, 2.4, 9.1, 8.3, 8.8

```

Zadanie 104 – ObiektRzeczywisty.java

Znajdź obiekt występujący w rzeczywistości i stwórz dla niego odpowiednią implementację w Javie (klasa plus przykładowe obiekty).

Zadanie 105 – NajlepsiGracze.java

Gry komputerowe posiadają listę najlepszych wyników osiągniętych przez poszczególnych graczy. Osoby, które uzyskały największą liczbę punktów zajmują najwyższe pozycje na liście. Utwórz program, który na podstawie osiągniętych rezultatów przez 30 graczy, sporządzi listę 10 najlepszych wyników. Przykładową listę przedstawiono poniżej. Liczbę uzyskanych punktów wygeneruj losowo z przedziału <0,10000>.

Tabela 2. Przykładowy ranking osób.

MIEJSCE	GRACZ	WYNIK
1	Terminator	3523
2	Alex	2865
3	Morfeusz	1982
4	Easyrider	1634
5	Merlin	1321

Rozwiązanie

Wykorzystaj w zadaniu poniższy kod programu. Klasa Gracz będzie przechowywać dane poszczególnych graczy (nazwa gracza, wynik), natomiast klasa NajlepszaDziesiątka listę dziesięciu graczy, którzy osiągnęli najwyższe rezultaty. Uzupełnij poniższe klasy wykorzystując enkapsulację danych, konstruktory oraz metody dostępne i modyfikujące.

```

class Gracz {
    String nick;
    int wynik;

    public Gracz(String nick, int wynik){
        this.nick = nick;
        this.wynik = wynik;
    }

    public int pobierzWynik(){
        return wynik;
    }

    public String toString(){
        return nick + " " + wynik + "\n";
    }
}

class NajlepszaDziesiątka {

    Gracz[] top;

    public NajlepszaDziesiątka(){
        top = new Gracz[10];
    }
}

```

```
    }

    public void dodajGracza(Gracz gracz){

    }

    public String toString() {
        String wynik = "";
        for(Gracz x : top) wynik += x;
        return wynik;
    }
}
```

Rozdział 6

Kompozycja i dziedziczenie

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

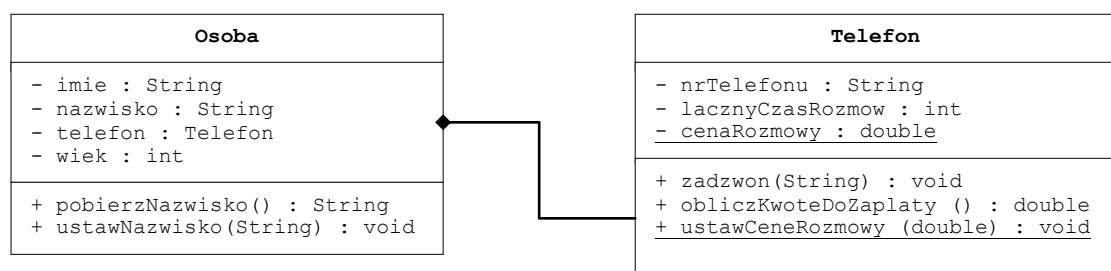
- wykorzystywać istniejący kod programu, w celu jego rozszerzenia,
- tworzyć nowe klasy pochodne, na podstawie klas już istniejących,
- operować składowymi klas nadrzędnych,
- grupować pliki aplikacji w formie pakietów.

Wprowadzenie do zagadnień

Idea programowania zorientowanego obiektowo umożliwia wielokrotne wykorzystanie istniejącego kodu programu. Dzięki temu możliwe jest użycie istniejących klas bez naruszania ich implementacji realizując tym samym ideę programowania przyrostowego. Podejście obiektowe w programowaniu dostarcza mechanizmów umożliwiających łatwe wyrażanie związków pojęć odnoszących się do modelowanej rzeczywistości. Najczęściej spotykane to kompozycja oraz dziedziczenie.

KOMPOZYCJA

Kompozycja⁵¹ wyraża relację „składa się z” lub „posiada”. W skład tworzonej klasy może wchodzić dowolna liczba obiektów, tworzonych na podstawie istniejących już klas. Ilustracją takich zależności jest najczęściej diagram klas UML.



Rys. 4. Relacja kompozycji.

Poniższy kod programu zawiera definicję klasy **Osoba** wykorzystującą jako pola obiekty dwóch innych klas, **String** oraz **Telefon**.

⁵¹ Kompozycja jest szczególnym przypadkiem agregacji i oznacza, iż obiekty składowe nie mogą istnieć bez obiektu głównego.

```

class Osoba {

    // deklaracja pól

    private String imie;        // kompozycja (wykorzystanie klasy String)
    private String nazwisko;    // kompozycja (wykorzystanie klasy String)
    private Telefon telefon;    // kompozycja (wykorzystanie klasy Telefon)
    private int wiek;

    // przykładowy konstruktor

    public Osoba(String imie, String nazwisko, Telefon telefon, int wiek){
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.telefon = telefon;
        this.wiek = wiek;
    }

    // przykładowe metody

    public String pobierzNazwisko() {
        return nazwisko;
    }

    public void ustawNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;
    }
}

```

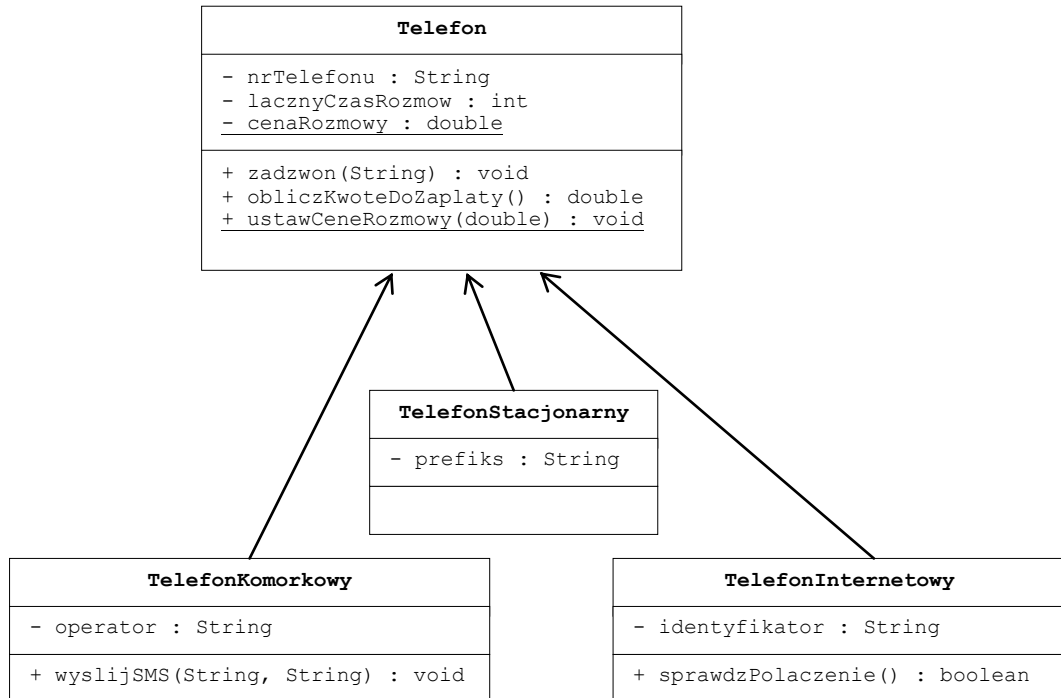
Proces tworzenia nowego obiektu klasy `Osoba` wraz z obiektami zależnymi (`Telefon`) został przedstawiony poniżej:

```
Osoba adam = new Osoba ("Adam", "Kowalski", new Telefon("823945321"), 21);
```

DZIEDZICZENIE

W celu określenia, iż jedno pojęcie jest uszczegółowieniem (lub uogólnieniem) innego, stosowane jest dziedziczenie. Wyraża ono relację „jest” i jest wykorzystywane wszędzie tam, gdzie należy wskazać, iż 2 pojęcia są do siebie podobne. Realizacja dziedziczenia polega na tym, iż nowo tworzona klasa przejmuje wszelkie cechy i zachowania z klas już istniejących, dodając bądź też modyfikując je, aby były one bardziej wyspecjalizowane. Dzięki temu powstaje nowa klasa określana terminem „podklasa” (ang. *subclass*) lub „klasa pochodna”, natomiast klasa, z której dokonano dziedziczenia to „nadklasa” (ang. *superclass*)⁵² lub „klasa bazowa”.

⁵² Używane są również pojęcia dziecka (ang. *child*) i rodzica (ang. *parent*) dla określania nazwy podklasy oraz klasy bazowej.



Rys. 5. Relacja dziedziczenia (klasa *Telefon* oraz klasy pochodne).

Dziedziczenie w języku Java realizowane jest za pomocą słowa kluczowego `extends`⁵³ po którym występuje nazwa klasy, z której dziedziczone są jej składowe:

```
class Student extends Osoba {
    // ciało klasy (składowe klasy)
}
```

Klasa *Student* rozszerza (dziedziczy) klasę *Osoba*. W tym przypadku *Osoba* jest klasą bazową, natomiast *Student* klasą pochodną, która dziedziczy wszystkie dostępne pola i metody klasy bazowej⁵⁴. W klasie pochodnej możliwe jest:

- deklarowanie nowych pól, które nie występują w klasie bazowej oraz tworzenie nowych pól o takiej samej nazwie, ukrywając zarazem pola oryginalne z klasy bazowej⁵⁵,
- deklarowanie metod, które nie występują w klasie bazowej oraz tworzenie nowych metod o takiej samej sygnaturze⁵⁶.

Rozszerzając możliwości przykładowej klasy *Telefon* można na jej podstawie utworzyć nowe klasy: *TelefonKomorkowy*, *TelefonStacjonarny* oraz *TelefonInternetowy*. Klasy te posiadają nowe cechy (np. `operator`, `prefiks`), jak i zachowania (np. `wyslij SMS`). Każda z klas posiada również

⁵³ Dziedziczenie w Javie jest jednokrotne (każda klasa dziedziczy tylko z jednej klasy). Klasy, które jawnie nie posiadają zadeklarowanego dziedziczenia (brak słowa kluczowego `extends` w nagłówku) domyślnie dziedziczą z klasy `Object`.

⁵⁴ Konstruktory nie są dziedziczone.

⁵⁵ Tworzenie w klasie pochodnej pól o takiej samej nazwie jak w klasie bazowej nie jest zalecane.

⁵⁶ Tworzenie w klasie pochodnej metod o takiej samej sygnaturze jak metody w klasie bazowej nazywane jest przesłanianiem metod (ang. *override*).

własną implementację metody `zadzwon(String)`. Tworząc nowe obiekty można posługiwać się również typem klasy bazowej (zarówno telefon stacjonarny jak i komórkowy jest przecież telefonem⁵⁷).

```
class TelefonKomorkowy extends Telefon {

    // nowe pole
    String operator;

    // konstruktor
    public TelefonKomorkowy (String numer, String operator) {
        super(numer); // wywołanie konstruktora klasy bazowej
        this.operator = operator;
    }

    // nowa metoda
    public void wyslijSMS(String nrTelefonu, String tresc) {
        System.out.println ("Wysylam SMS'a o tresci: " + tresc);
        System.out.println ("pod numer: " + nrTelefonu);
    }

    // nowa implementacja metody zadzwon (metoda przesłonięta)
    public void zadzwon(String nrTelefonu) {
        System.out.println ("Dzwonie z komórki do: " + nrTelefonu);
    }
}

class TelefonStacjonarny extends Telefon {

    // nowe pole
    String prefiks;

    // konstruktor
    public TelefonStacjonarny (String numer, String prefiks) {
        super(numer);
        this.prefiks = prefiks;
    }

    // nowa implementacja metody zadzwon (metoda przesłonięta)
    public void zadzwon(String nrTelefonu) {
        System.out.println ("Dzwonie ze stacjonarnego do: " + nrTelefonu);
    }
}

class TelefonInternetowy extends Telefon {

    // nowe pole
    String identyfikator;

    // konstruktor
    public TelefonInternetowy (String numer, String identyfikator) {
        super(numer);
        this.identyfikator = identyfikator;
    }

    // nowa metoda
    public boolean sprawdzPolaczenie() {
        return true;
    }

    // nowa implementacja metody zadzwon (metoda przesłonięta)
    public void zadzwon(String nrTelefonu) {
        if(sprawdzPolaczenie())
            System.out.println ("Dzwonie z internetowego do: " + nrTelefonu)
        else
            System.out.println ("Brak polaczenia z internetem");
    }
}

public class Test {

    public static void main(String args[]) {

        Telefon telefonJasia = new Telefon("867312632");
```

⁵⁷ Taki sposób tworzenia obiektów nazywany jest rzutowaniem w górę (ang. *upcasting*).

```

        Telefon komorkaMarka = new TelefonKomorkowy("606345956", "Era");
        Telefon stacjonarnyKasi = new TelefonStacjonarny("125439876", "1044");
        Telefon internetowyAni = new TelefonInternetowy("146743253", "Anula7");
    }
}

```

POLIMORFIZM

Zarówno kompozycja, jak i dziedziczenie, które pozwala kojarzyć klasy obiektów w hierarchie klas, należą do fundamentalnych własności podejścia obiektowego. Umożliwiają realizację idei programowania przyrostowego oraz późniejszą szybką modyfikację kodu programu.

Tworząc obiekty na podstawie klas dziedziczących z innych klas, można jako zmienną referencyjną podać typ klasy bazowej⁵⁸. Wywołując następnie odpowiednie metody kompilator Javy decyduje, która z nich ma zostać wykonana (na podstawie typu obiektu, na rzecz którego są one wywołane). Taki sposób wywołania metod nazywany jest polimorfizmem.

Poniższy przykład przedstawia tablicę elementów klasy `Telefon`. Poszczególne elementy tablicy zawierają referencje zarówno do obiektów klasy `Telefon`, jak również do obiektów utworzonych na podstawie klas pochodnych (`TelefonKomorkowy`, `TelefonStacjonarny`).

```

Telefon[] tablicaTelefonow = new Telefon[4];

tablicaTelefonow[0] = new Telefon("634295432");
tablicaTelefonow[1] = new TelefonKomorkowy("504295432", "Orange");
tablicaTelefonow[2] = new TelefonStacjonarny("126493042", "1033");
tablicaTelefonow[3] = new TelefonInternetowy("325649344", "lech23");

```

Wywołanie dla każdego obiektu znajdującego się w tablicy `tablicaTelefonow` metody `zadzwon("112")` będzie wywołaniem polimorficznym. W zależności od klasy obiektu wywołana zostanie odpowiednia metoda.

```

for (Telefon tel : tablicaTelefonow)
    tel.zadzwon("112");

```

PAKIETY

Powiązane ze sobą klasy i interfejsy⁵⁹ umieszczane są w jednostkach bibliotecznych zwanych pakietami (ang. *package*). Takie grupowanie pozwala na szybsze odnalezienie właściwych klas i interfejsów, jak również wpływa korzystnie na przejrzystość pisanych aplikacji. Dodatkowe zalety pakietów to m.in.:

- jasne określenie związków (powiązań) między klasami,
- unikanie konfliktu nazw⁶⁰,
- możliwość skorzystania z dodatkowych modyfikatorów dostępu⁶¹.

Pakiety tworzy się poprzez użycie w kodzie źródłowym programu słowa kluczowego `package`⁶², po którym występuje nazwa pakietu⁶³. Określa to lokalizację pliku skompilowanego (`.class`), który musi zostać umieszczony w podanym pakiecie (folderze)⁶⁴:

```

package telefonia;
package uczelnia.podstawy.narzedzia;

```

Pakiety posiadają podobną strukturę, jak katalogi w systemie plików (mogą być również zagnieżdżane). Utworzenie pakietu `uczelnia.podstawy.narzedzia` jest równoznaczne z utworzeniem struktury katalogów na dysku `./uczelnia/podstawy/narzedzia`. Wszystkie pliki należące do tego pakietu

⁵⁸ Można użyć również nazwy interfejsu, który dana klasa implementuje.

⁵⁹ Tematyka interfejsów zostanie poruszona w następnym rozdziale.

⁶⁰ Pakiety umożliwiają korzystanie z klas lub interfejsów o takiej samej nazwie, ale innej strukturze (np. identyczna nazwa klas w odrębnych pakietach).

⁶¹ Możliwość użycia modyfikatora `package`.

⁶² Słowo `package` musi pojawić się jako pierwsza instrukcja programu.

⁶³ Nazwy pakietów zawierają tylko małe litery i są tworzone na kilka sposobów (np. wykorzystując odwrotny zapis domeny internetowej firmy).

⁶⁴ Należy pamiętać, iż spoza pakietu dostęp możliwy jest tylko do elementów publicznych danego pakietu (klasy, interfejsy). Klasy, które nie określają przynależności do konkretnego pakietu są grupowane w tak zwany pakiet domyślny, nie posiadający żadnej nazwy.

powinny znaleźć się w przytoczonej lokalizacji. Korzystanie z elementów pakietu umożliwia słowo kluczowe `import`, po którym występuje nazwa pakietu oraz nazwa klasy, bądź też znak gwiazdki symbolizujący wszystkie klasy należące do wskazanego pakietu⁶⁵.

```
// wszystkie składowe pakietów telefonia oraz uczelnia.podstawy.narzedzia są dostępne
import telefonia.*;
import uczelnia.podstawy.narzedzia.*;
```

Nazwa pakietów dostarczanych wraz z Javą rozpoczyna się od słowa `java` (pakiety związane ze specyfikacją języka) bądź też `javax` (rozszerzenie specyfikacji).

Pojęcia dodatkowe

KLASA OBJECT

`Object`⁶⁶ jest główną klasą, z której pośrednio lub bezpośrednio dziedziczą wszystkie klasy. Zawiera ona szereg metod ułatwiających operacje na obiektach. Najważniejsze z nich to⁶⁷:

- `clone()` – tworzy kopię obiektu,
- `toString()` – zwraca reprezentację obiektu w formie łańcucha tekstowego,
- `equals(Object)` – porównuje dwa obiekty,
- `getClass()` – zwraca nazwę klasy na podstawie, której powstał obiekt.

Najczęściej używane to `toString()` oraz `equals(Object)`, które przesłaniane są w klasach pochodnych.

OPERATOR INSTANCEOF

Operator `instanceof` służy do sprawdzenia czy dany obiekt należy do wskazanej klasy. Przyjmuje on wartość `true` lub `false`. Wartość `true` oznacza, iż dany obiekt należy do wskazanej klasy lub też dowolnej podklasy.

```
boolean sprawdzenie = telefonAni instanceof Telefon;
```

SŁOWO KLUCZOWE SUPER

Dostęp do składowych klasy bazowej z klasy dziedziczącej możliwy jest za pomocą słowa kluczowego `super`. W ten sposób można również odwoływać się do konstruktorów klasy nadrzędnej⁶⁸ (nie są one dziedziczone). Wywołanie konstruktora klasy bazowej musi być pierwszą instrukcją w ciele konstruktora podklasy.

Pytania sprawdzające

1. Czym jest kompozycja w programowaniu obiektowym?
2. Wskaż różnice pomiędzy pojęciami klasa bazowa, a podklasa.
3. Jakie składowe klasy bazowej dziedziczy podklasa?
4. Czy każda klasa w Javie dziedziczy z innej klasy? Czy możliwe jest dziedziczenie z wielu klas?
5. Wyjaśnij pojęcie polimorfizmu.
6. Jakie zadania spełnia klasa `Object`? Jakimi metodami dysponuje?
7. Podaj przykłady użycia metody `toString()` z klasy `Object`?
8. Jaką funkcję pełni operator `instanceof`?

⁶⁵ Kompilator Javy poszukuje klas wykorzystując ścieżkę zbudowaną ze zmiennej środowiskowej `CLASSPATH` oraz nazwy konkretnego pakietu (kropki w pakiecie zostają zastąpione znakiem `/`).

⁶⁶ Klasa `Object` jest jedyną klasą nie posiadającą klasy bazowej. Znajduje się ona na szczycie hierarchii klas.

⁶⁷ Pozostałe metody związane są z synchronizacją wątków, generowaniem unikalnych kluczy dla obiektów oraz czyszczeniem pamięci przy usuwaniu obiektów.

⁶⁸ Warto pamiętać o sytuacji, gdy konstruktor podklasy nie wywołuje jawnie konstruktora z klasy bazowej. W takich sytuacjach Java automatycznie wywoła konstruktor domyślny (bezparametrowy) klasy bazowej.

9. Wskaż różnice pomiędzy słowami kluczowymi `this` i `super`.
10. Określ zalety używania pakietów.
11. W jaki sposób możliwe jest użycie klas znajdujących się we wskazanym pakiecie?
12. Czy możliwe jest przesłonięcie metody `equals` (`Object`)? Jeśli tak, to jakie zadanie będzie spełniać ta metoda?

Zadania do wykonania

KOMPOZYCJA

Zadanie 106 – *Pojazd.java*

Pojazd jest środkiem transportu ułatwiającym przemieszczanie się. Aby było to możliwe, musi posiadać napęd (silnik). Utwórz uproszczony model pojazdu wraz z silnikiem.

Rozwiązanie

Poniższy kod programu zawiera klasę reprezentującą samochód oraz silnik pojazdu. Zwróć uwagę na relacje zachodzące pomiędzy występującymi klasami. Jaki to typ relacji?

```
public class Pojazd {
    private String nazwa;
    private Silnik silnik;

    public Pojazd(String nazwa, String typSilnika) {
        this.nazwa = nazwa;
        silnik = new Silnik(typSilnika);
    }

    public void ruszaj(){
        System.out.println("Wsiadam do mojego " + nazwa);
        silnik.zapal();
        System.out.println("Ruszam - gaz do dechy... i w drogę...");
    }

    public void zatrzymaj(){
        System.out.println("Zatrzymuję się z piskiem opon...");
        silnik.zgas();
        System.out.println("Wychodzę z samochodu...");
    }

    public static void main(String[] args) {
        Pojazd pojazd = new Pojazd("Audi Quattro", "Turbo Diesel");
        pojazd.ruszaj();
        pojazd.zatrzymaj();
    }
}

class Silnik {
    private String typ;

    public Silnik(String typ){
        this.typ = typ;
    }

    public void zapal(){
        System.out.println("Zapalam silnik..." + typ);
    }

    public void zgas(){
        System.out.println("Gaszę silnik...");
    }

    public String pobierzTyp(){
        return typ;
    }
}
```

Zadanie 107 – Restauracja.java

Restauracja „Europa” składa się z dwóch sal – dla palących oraz niepalących. Każda sala mieści określoną liczbę gości i wyposażona jest w stoliki (stół plus krzesła) oraz telewizor. Dodatkowo sala dla palących posiada stół do snookera. Wykorzystując Javę, utwórz model restauracji oraz wszystkie jej elementy składowe. Utwórz przykładowy stan sal (sala 1 – 48 miejsc, sala 2 – 32 miejsca). Dodaj losowo gości znajdujących się w poszczególnych salach. Wyświetl na konsoli informacje prezentujące stan rzeczywisty restauracji (liczba gości w każdej sali, liczba wolnych stolików, stan zajętości stołu do snookera, nazwa programu aktualnie nadawanego w TV). Wykorzystaj kompozycję oraz poznane pojęcia.

Wskazówka

Utwórz klasy: *Restauracja*, *Sala*, *Stolik*, *Stol*, *Krzeslo*, *Telewizor*, *Snooker*, *Gosc*

DZIEDZICZENIE I POLIFORMIZM*Zadanie 108 – Zwierzeta.java*

Na podwórzu gospodarskim biegają zwierzęta domowe (pies, kot, krowa), wydające odgłosy. Przedstaw przy pomocy języka Java zaistniałą sytuację.

Rozwiązanie

Poniższy kod programu zawiera rozwiązanie zadania. Zwróć uwagę na relacje zachodzące pomiędzy klasami. Czy potrafisz je scharakteryzować? W jaki sposób wywoływana jest metoda `dajGlos()`?

```
public class Zwierzeta {
    public static void main(String[] args) {
        Zwierze[] zwierzeta = new Zwierze[3];
        zwierzeta[0] = new Pies();
        zwierzeta[1] = new Kot();
        zwierzeta[2] = new Krowa();
        System.out.println("Rykwisko...");
        for (Zwierze zwierze: zwierzeta) {
            zwierze.dajGlos();
        }
    }
}

class Zwierze {
    public void dajGlos() {
        System.out.println("Głos zwierzęcia");
    }
}

class Pies extends Zwierze {
    public void dajGlos() {
        System.out.println("hau hau");
    }
}

class Kot extends Zwierze {
    public void dajGlos() {
        System.out.println("miau miau");
    }
}

class Krowa extends Zwierze {
    public void dajGlos() {
        System.out.println("muuuuu");
    }
}
```

Zadanie 109 – MiniZoo.java

Rodzice Kornela stworzyli na swojej posiadłości małe zoo. Aktualnie przebywają tam pandy, małpy oraz kucyki. Teo to 3 letni miś panda. W zoo mieszka wraz ze swoimi małymi 5 miesięcznymi synkami Pikiem i Mikiem. Romi i Tysia to dwie 4-letnie małpki. Inwentarz dopełniają 3 kucyki, 2 letni Gatek, 3 letni Kajtek i 5 letni Taksel. 3 razy dziennie wszystkie zwierzęta otrzymują pokarm. W każdej chwili możliwe jest wyznaczenie liczebności wszystkich zwierząt. Jeśli zoo przyjmuje nowe zwierzę (lub też rodzi się jakieś), liczba zwierząt zwiększa się. Utwórz reprezentację zoo w postaci

odpowiednich klas. Stwórz obiekty odzwierciedlające każdego mieszkańca, jak i całe zoo. Przedstaw w postaci metod wieczorne karmienie zwierząt oraz narodziny nowego mieszkańca zoo (małpki Igi). Wyświetl wszystkie zwierzęta wraz z ich wiekiem. Wykorzystaj dziedziczenie oraz polimorfizm.

Wskazówka

Niezbędne klasy: *MiniZoo*, *Zwierze*, *Panda*, *Malpa*, *Kucyk*.

Zadanie 110 – *Punkty.java*, *Punkt.java*

Napisz program umożliwiający określanie oraz zmianę lokalizacji (przesuwanie) punktów na płaszczyźnie.

Rozwiązanie

Utwórz klasę *Punkt*, opisującą dowolny punkt na płaszczyźnie. Zapisz ją w odrębnym pliku źródłowym.

```
public class Punkt {

    private int x, y; // współrzędne punktu

    public Punkt(int px, int py){
        x = px;
        y = py;
    }

    public int pobierzX(){
        return x;
    }

    public int pobierzY(){
        return y;
    }

    public void ustawX(int px){
        x = px;
    }

    public void ustawY(int py){
        y = py;
    }

    public void przesun(int dx, int dy){
        x += dx;
        y += dy;
    }

    public String toString(){
        return String.format("(%d,%d)", x, y);
    }
}
```

Zasadnicza część programu (w tym metoda *main()*) zawarta jest w klasie *Punkty*. Dokonując kompilacji programu wystarczy skompilować kod zawarty w pliku *Punkty.java*. Ze względu na występujące odwołania, kompilacja pliku *Punkt.java* zostanie przeprowadzona automatycznie.

```

public class Punkty {
    public static void main(String[] args) {
        Punkt punktA = new Punkt(3, 7);
        Punkt punktB = new Punkt(2, 11);
        Punkt punktC = new Punkt(6, 5);
        System.out.print("Współrzędne punktów: ");
        System.out.printf("%s %s %s\n", punktA, punktB, punktC);
        System.out.println("Zmiana położenia punktów...");
        punktA.przesun(2, -1);
        punktB.przesun(2, -1);
        punktC.przesun(2, -1);
        System.out.print("Współrzędne punktów: ");
        System.out.printf("%s %s %s\n", punktA, punktB, punktC);
    }
}

```

Zadanie 111 – Odcinki.java, Punkt.java

Napisz program określający położenie odcinka na płaszczyźnie. Program powinien umożliwiać również wyznaczanie długości odcinka.

Rozwiązanie

Wykorzystaj poprzednio zdefiniowaną klasę Punkt.

```

public class Odcinki {
    public static void main(String[] args) {
        Odcinek odcinek = new Odcinek( new Punkt(2,3), new Punkt(7,4) );
        System.out.println(odcinek);
        System.out.println("Zmiana położenia odcinka...");
        odcinek.przesun(-1, 4);
        System.out.println(odcinek);
    }
}

class Odcinek {

    private Punkt punktA, punktB; // współrzędne końców odcinka

    public Odcinek(Punkt ppunktA, Punkt ppunktB){
        punktA = ppunktA;
        punktB = ppunktB;
    }

    public double dlugosc(){
        int xpunktA = punktA.pobierzX();
        int ypunktA = punktA.pobierzY();
        int xpunktB = punktB.pobierzX();
        int ypunktB = punktB.pobierzY();
        return Math.sqrt( (xpunktA-xpunktB)*(xpunktA-xpunktB) +
                           (ypunktA-ypunktB)*(ypunktA-ypunktB) );
    }

    public void przesun(int dx, int dy){
        punktA.przesun(dx, dy);
        punktB.przesun(dx, dy);
    }

    public String toString(){
        String opis;
        opis = String.format("Odcinek: A%s,B%s,", punktA, punktB);
        opis += String.format(" długość %3.1f", dlugosc());
        return opis;
    }
}

```

Zadanie 112 – Rysunek.java, Figura.java, Kolo.java, Prostokat.java, Punkt.java

Napisz program, który umożliwi umieszczanie figur geometrycznych na płaszczyźnie.

Rozwiązanie

Wykorzystaj uprzednio zdefiniowaną klasę Punkt.

```

public class Rysunek {

    int liczbaFigur = 0;
}

```



```

Figura[] figury;

public Rysunek(int pmaxLiczbaFigur){
    figury = new Figura[pmaxLiczbaFigur];
}

public void dodajFigure(Figura nowaFigura){
    figury[liczbaFigur++] = nowaFigura;
}

public void przesun(int dx, int dy){
    for(int i=0; i<liczbaFigur; i++)
        figury[i].przesun(dx, dy);
}

public String toString(){
    String s = "";
    for(int i=0; i<liczbaFigur; i++)
        s += figury[i] + "\n";
    return s;
}

public static void main(String[] args) {
    Rysunek rysunek = new Rysunek(8);
    rysunek.dodajFigure(new Kolo(new Punkt(2, 5), 7));
    rysunek.dodajFigure(new Prostokat(
        new Punkt(5, 2), new Punkt(5, 4),
        new Punkt(1, 4), new Punkt(1, 2) ));
    rysunek.dodajFigure(new Kolo(new Punkt(4, 1), 3));
    rysunek.dodajFigure(new Kolo(new Punkt(8, 9), 1));
    rysunek.dodajFigure(new Prostokat(
        new Punkt(3, 0), new Punkt(4, 0),
        new Punkt(4, 8), new Punkt(3, 8) ));
    System.out.println(rysunek);
    System.out.println("Przesunięcie figur...");
    rysunek.przesun(3, -1);
    System.out.println(rysunek);
}

}

class Figura {
    public void przesun(int dx, int dy) {
    }
    public double obliczPole() {
        return 0.0;
    }
    public double obliczObwod() {
        return 0.0;
    }
}

class Kolo extends Figura {

    private Punkt punktA; // środek koła
    private int r; // promień

    public Kolo(Punkt ppunktA, int pr){
        punktA = ppunktA;
        r = pr;
    }

    public double obliczPole(){
        return Math.PI * r * r;
    }

    public double obliczObwod(){
        return 2 * Math.PI * r;
    }

    public void przesun(int dx, int dy){
        punktA.przesun(dx, dy);
    }

    public String toString(){
        return String.format("Koło: A%s, r=%d, pole=%f, obwód=%f",

```

```

        punktA, r, obliczPole(), obliczObwod() );
    }
}

class Prostokat extends Figura {

    private Punkt punktA, punktB, punktC, punktD; // współrzędne wierzchołków

    public Prostokat(Punkt ppunktA, Punkt ppunktB, Punkt ppunktC, Punkt ppunktD) {
        punktA = ppunktA; punktB = ppunktB; punktC = ppunktC; punktD = ppunktD;
    }

    // odległość pomiędzy 2 punktami
    private double obliczXY(Punkt pX, Punkt pY){
        int xpunktA = pX.pobierzX();
        int ypunktA = pX.pobierzY();
        int xpunktB = pY.pobierzX();
        int ypunktB = pY.pobierzY();
        return Math.sqrt( (xpunktA-xpunktB)*(xpunktA-xpunktB) +
            (ypunktA-ypunktB)*(ypunktA-ypunktB) );
    }

    public double obliczPole(){
        double a = obliczXY(punktA, punktB);
        double b = obliczXY(punktA, punktD);
        return a*b;
    }

    public double obliczObwod(){
        double a = obliczXY(punktA, punktB);
        double b = obliczXY(punktA, punktD);
        return 2*a + 2*b;
    }

    public void przesun(int dx, int dy){
        punktA.przesun(dx, dy);
        punktB.przesun(dx, dy);
        punktC.przesun(dx, dy);
        punktD.przesun(dx, dy);
    }

    public String toString(){
        return String.format("Prostokat: A%s, B%s, C%s, D%s ",
            punktA, punktB, punktC, punktD) +
            String.format("pole=%f, obwód=%f", obliczPole(), obliczObwod());
    }
}

```

Zadanie 113 – *Trojkat.java, Trapez.java, Romb.java*

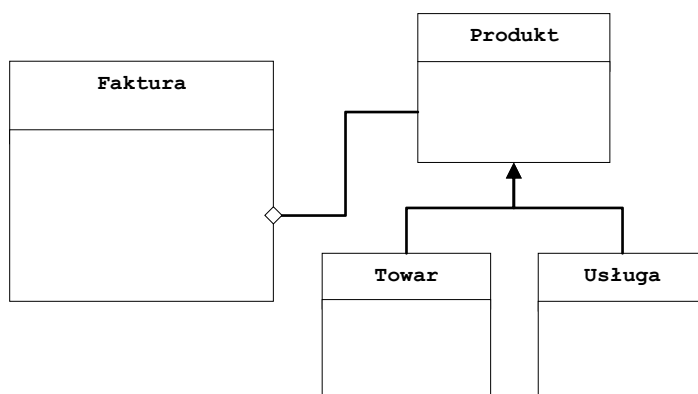
Uzupełnij program *Rysunek.java* o możliwość umieszczania na płaszczyźnie kolejnych figur geometrycznych (romb, trapez oraz trójkąt).

Zadanie 114 – *KreatorFakturVAT.java*

Firma Handlowa-Usługa MORINEX sp. z o.o. wdraża system komputerowy wspomagający prowadzenie działalności gospodarczej. Jednym z jego elementów jest oprogramowanie usprawniające wystawianie faktur VAT. Napisz program, który realizował będzie podaną funkcjonalność.

Rozwiązanie

Faktura VAT zawiera następujące informacje: numer faktury, data sprzedaży, dane sprzedawcy (nazwa, adres, kod pocztowy, miejscowość, NIP), dane nabywcy (nazwa, adres, kod pocztowy, miejscowość, NIP), wykaz sprzedanych produktów (towarów i/lub usług), łączna kwota brutto do zapłaty. W skład listy produktów wchodzi towary lub usługi. Każdy towar charakteryzuje: nazwa towaru, jednostka miary, ilość, cena jednostkowa netto, wartość netto, stawka VAT (3, 7 lub 22%), wartość brutto. W przypadku usług są to: nazwa usługi, liczba roboczogodzin, cena roboczogodziny, wartość netto, stawka VAT (7 lub 22%), wartość brutto. Ogólny schemat występujących zależności przedstawiony został na poniższym diagramie UML:



Rys. 6. Graficzna reprezentacja zależności między klasami.

Natomiast przykładową postać faktury przedstawiono poniżej:

FAKTURA VAT nr 12321
Data sprzedaży: 17-04-2009

SPRZEDAWCA:
Firma Usługowo-Handlowa MORINEX
ul. Koszykowa 15, 30-194 Kraków
NIP 677-001-20-12

NABYWCA:
Firma VENPOL SA
ul. Długa 8/7, 31-723 Kraków
NIP 982-10-32-203

Nazwa towaru/usługi	j.m.	cena	ilość	wart.netto	st.VAT	wart.brutto
Kalosze męskie	szt	12,30	3	36,90	22	45,02
Cement	kg	3,80	230	874,00	22	1066,28
Montaż instalacji	godz.	38,00	7	266,00	7	284,62
...						

Razem do zapłaty: 1395,92 zł

METODY KLASY OBJECT

Zadanie 115 – PorównanieObiektow.java

System komputerowy dziekanatu uczelni zawiera dane studentów, którzy rozróżniani są na podstawie numeru albumu. Warunkiem koniecznym jest zapewnienie unikalności numerów (każdy student powinien posiadać różny nr albumu). Zaprojektuj prostą klasę reprezentującą studenta (pola: imię, nazwisko, nr albumu). Utwórz przykładowych studentów o różnych i takich samych numerach albumu. Sprawdź, jaką wartość zwraca porównanie obiektów zawierających taki sam numer albumu. Wykorzystaj przesłanianie metody equals (Object) do porównania obiektów oraz toString () do ich wyświetlania.

TWORZENIE PAKIETÓW

Zadanie 116 – TelefonKasi.java

Grupowanie klas realizujących zbliżoną funkcjonalność ułatwia tworzenie aplikacji. Utwórz pakiet telefonia grupujący wszystkie klasy dotyczące telefonów. Wykorzystaj pakiet importując z niego klasy potrzebne do utworzenia telefonu komórkowego Kasi (o numerze 503984532 w sieci Plus) oraz wyślij z niego SMS o treści "Zadanie z pakietem wykonane" na numer 602875295.

Rozwiązanie

Klasy wchodzące w skład pakietu `telefon` powinny być klasami publicznymi⁶⁹. Niezbędne jest użycie słowa kluczowego `package` wraz z nazwą pakietu jako pierwszej instrukcji zawartej w pliku źródłowym.

```
package telefon; // pierwsza instrukcja w pliku źródłowym

public class Telefon {
    // ciało klasy (składowe klasy)
}

package telefon;

public class TelefonKomorkowy extends Telefon {
    // ciało klasy (składowe klasy)
}

package telefon;

public class TelefonStacjonarny extends Telefon {
    // ciało klasy (składowe klasy)
}

package telefon;

public class TelefonInternetowy extends Telefon {
    // ciało klasy (składowe klasy)
}

import telefon.*;

public class TelefonKasi {
    TelefonKomorkowy telefonKasi = new TelefonKomorkowy ("503984532", "Plus");
    telefonKasi.wyslijSMS("602875295", "Zadanie z pakietem wykonane");
}
```

Zadanie 117 – FlotaPojazdow.java

Flota pojazdów firmy SPEED składa się samochodów ciężarowych oraz osobowych. Wśród tych ostatnich występują samochody służbowe oraz prywatne. Liczebność na dzień 17-04-2009 przedstawiała się następująco:

- 4 wywrotki Scania
- 2 ciągniki siodłowe MAN
- 2 samochody Mercedes (służbowe)
- 3 samochody Fiat Punto (służbowe)
- 5 samochodów Ford Fiesta (prywatnych, użyczonych na potrzeby firmy)

Zaprojektuj klasy odpowiadające strukturze floty pojazdów w firmie SPEED wraz z przykładowymi właściwościami. W zależności od kategorii, umieść klasy w oddzielnych pakietach. Utwórz obiekty wg podanej specyfikacji.

⁶⁹ Nazwa klasy publicznej musi być zgodna z nazwą pliku źródłowego. W przykładowym rozwiązaniu każda z klas publicznych musi znajdować się w oddzielnym pliku źródłowym.

Rozdział 7

Interfejsy i klasy abstrakcyjne

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

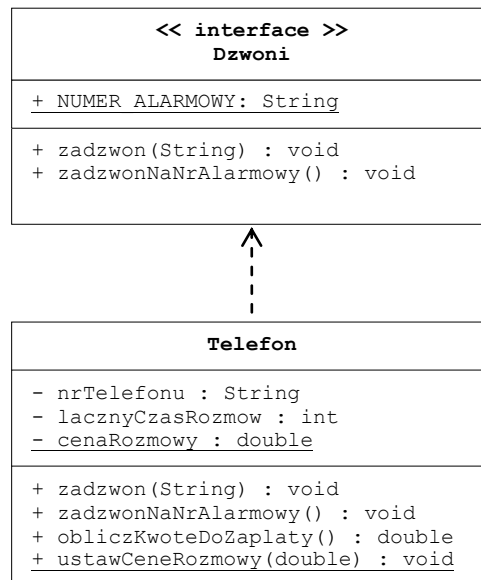
- rozszerzać funkcjonalność klas poprzez wykorzystanie interfejsów,
- konstruować i efektywnie wykorzystywać klasy abstrakcyjne,
- tworzyć programy z użyciem klas wewnętrznych oraz finalnych.

Wprowadzenie do zagadnień

Niewątpliwym atutem programowania obiektowego jest zbliżenie tworzonych programów do sposobu postrzegania rzeczywistości przez człowieka. Przy pomocy klas możliwe jest modelowanie pojęć z dziedziny tworzonego programu. Efektywne wykorzystanie narzędzi stosowanych w programowaniu zorientowanym obiektowo (interfejsy, klasy abstrakcyjne, wewnętrzne oraz finalne) pozwala na uzyskanie wysokiej elastyczności w procesie modelowania.

INTERFEJSY

Interfejsy w programowaniu obiektowym stanowią zbiór wymagań dotyczących klas, które będą go stosować. Wyrażają sposób opisu funkcjonalności klasy bez określania, w jaki sposób będzie to uzyskane. W skład interfejsów wchodzi metody oraz pola. Deklaracja metod składa się z nagłówek (brak jest ciała metody), natomiast pola interfejsu to wyłącznie stałe statyczne z jawnie określoną wartością. W deklaracjach wszystkich składowych niezbędne jest użycie modyfikatora `public`.



Rys. 7. Interfejs i klasa implementująca.

Klasy wykorzystujące interfejs nie rozszerzają jego możliwości, ale go implementują. Oznacza to, że klasa musi zawierać definicję wszystkich metod (ciała tych metod) zadeklarowanych w interfejsie. Składnia interfejsu zbliżona jest do deklaracji klasy. W miejsce `class` stosowane jest słowo kluczowe `interface`⁷⁰.

```

interface Dzwoni {
    public static String NUMER_ALARMOWY = "112";

    public void zadzwon(String);
    public void zadzwonNaNrAlarmowy();
}
  
```

Implementacja interfejsu polega na umieszczeniu słowa kluczowego `implements` w nagłówku deklaracji klasy, a następnie wymienieniu nazwy interfejsu, który klasa implementuje⁷¹. Składowymi klasy stają się wszelkie pola i metody występujące w definicji klasy oraz pola i metody określone w deklaracji interfejsu. Poniższy kod programu ilustruje implementację interfejsu `Dzwoni`.

```

class Telefon implements Dzwoni {

    // deklaracja pól
    private String numerTelefonu;
    private int lacznyCzasRozmow;
    private static double cenaRozmowy = 0.48;    // zł/min.

    // konstruktor
    public Telefon (String numer) {
        numerTelefonu = numer;
    }

    // deklaracja metod
    public double obliczKwoteDoZaplaty() {
        return cenaRozmowy * (lacznyCzasRozmow / 60);
    }
}
  
```

⁷⁰ Dopuszczalne jest dziedziczenie interfejsów.

⁷¹ Możliwa jest implementacja dowolnej liczby interfejsów przez pojedynczą klasę co pozwala na realizację tzw. dziedziczenia wielobazowego.

```

    public static void ustawCeneRozmowy(double nowaCena) {
        cenaRozmowy = nowaCena;
    }

    // metody wymagane przez interfejs
    public void zadzwon(String nrTelefonu) {
        System.out.println ("Dzwonię do: " + nrTelefonu);
        System.out.println ("Dryń, dryń...");
        System.out.println ("Rozmowa w toku...");
        int czasRozmowy = (int) (Math.random()*3600);
        lacznyCzasRozmow += czasRozmowy;
        System.out.println ("Rozmowa zakończona. ");
        System.out.printf ("Czas rozmowy: %d min. %d sek.",
            czasRozmowy/60, czasRozmowy%60);
    }

    public void zadzwonNaNrAlarmowy() {
        System.out.println ("Dzwonię do: " + Dzwoni.NUMER_ALARMOWY);
        System.out.println ("Dryń, dryń...");
        System.out.println ("Centrum pomocy, słucham");
    }
}

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

        Telefon telefonKasi = new Telefon("1276594633");

        telefonKasi.zadzwon("606342765");
        telefonKasi.zadzwonNaNrAlarmowy();
    }
}

```

Interfejsy, podobnie jak klasy, mogą być stosowane jako typ danych przy deklaracji zmiennej⁷². Jej wartość stanowi odwołanie do obiektu dowolnej klasy, która implementuje interfejs⁷³.

KLASY I METODY ABSTRAKCYJNE

Klasy abstrakcyjne wykorzystywane są w procesie modelowania zawierając jedynie ogólne cechy i zachowania obiektów. Nie wszystkie cechy i zachowania muszą zostać szczegółowo opisane. Część z metod (nawet wszystkie⁷⁴) może składać się wyłącznie z deklaracji nazwy (tzw. metody abstrakcyjne⁷⁵) poprzedzonej identyfikatorem `abstract`. Umożliwia to stworzenie ogólnego modelu bez konieczności zagłębiania się w szczegóły implementacyjne. Klasa abstrakcyjna nie może stanowić podstawy dla utworzenia obiektu, możliwe jest natomiast na jej podstawie tworzenie klas pochodnych, które muszą zaimplementować wszystkie metody abstrakcyjne występujące w klasie bazowej. Deklaracja klasy abstrakcyjnej odbywa się za pomocą słowa kluczowego `abstract` umieszczonego przed nazwą klasy.

```

abstract class Figura {
    protected String kolor;

    public Figura(String kolor) {
        this.kolor = kolor;
    }

    public abstract double obliczPowierzchnie();
}

class Kwadrat extends Figura {
    private double bok;

    public Kwadrat(String kolor, double bok) {
        super(kolor);
        this.bok = bok;
    }

    public double obliczPowierzchnie() {

```

⁷² Jest to tak zwane rzutowanie rozszerzające.

⁷³ Odwołania do metod, które nie występują w interfejsie możliwe są za pomocą odpowiedniego rzutowania (tzw. rzutowanie zawężające).

⁷⁴ W przypadku, gdy klasa składa się wyłącznie z metod abstrakcyjnych, preferowane jest stosowanie interfejsów.

⁷⁵ Klasa posiadająca jakąkolwiek metodę abstrakcyjną staje się automatycznie klasą abstrakcyjną (jej nazwa musi zostać poprzedzona identyfikatorem `abstract`).

```

        return bok * bok;
    }
}

class Prostokat extends Figura {
    private double szerokosc;
    private double wysokosc;

    public Prostokat(String kolor, double szerokosc, double wysokosc) {
        super(kolor);
        this.szerokosc = szerokosc;
        this.wysokosc = wysokosc;
    }
    public double obliczPowierzchnie() {
        return szerokosc * wysokosc;
    }
}

```

KLASY I METODY FINALNE

Klasy, z których nie jest możliwe dziedziczenie nazywane są klasami finalnymi, co wyrażane jest za pomocą słowa kluczowego `final`⁷⁶.

```

public final class Procesor {
    // ciało klasy (składowe klasy)
}

```

W przypadku metod, użycie słowa kluczowego `final` powoduje, że nie jest możliwe jej przesłonięcie w klasach dziedziczących.

KLASY WEWNĘTRZNE

W przypadku ograniczonego użycia tworzonej klasy preferowane jest zdefiniowanie jej jako klasy wewnętrznej (możliwe jest ukrycie jej implementacji). Definicja klasy wewnętrznej musi znaleźć się w ciele innej klasy⁷⁷.

```

class Testowa {

    // ciało klasy głównej (składowe klasy)

    class KlasaWewnetrzna {
        // ciało klasy wewnętrznej (składowe klasy)
    }
}

```

Klasa wewnętrzna posiada dostęp do wszystkich składowych klasy w której została zdefiniowana⁷⁸. Nie ma również żadnych ograniczeń co do wykorzystania dziedziczenia, czy też implementacji interfejsów. Szczególnym przypadkiem klas wewnętrznych są klasy anonimowe⁷⁹ (nie posiadające nazwy).

Pytania sprawdzające

1. Określ cel i zastosowanie interfejsów.
2. Czy specyfikacja języka Java narzuca ograniczenia na liczbę interfejsów, jakie może implementować tworzona klasa?
3. Czy możliwe jest tworzenie interfejsów pochodnych?
4. W jakim przypadku klasa implementująca interfejs nie musi tworzyć wszystkich metod występujących w tym interfejsie?

⁷⁶ Przykładami klas finalnych są również klasy opakowujące, czy też klasa `String`.

⁷⁷ Klasy wewnętrzne mogą być również definiowane wewnątrz metod. Tak utworzone klasy noszą nazwę lokalnych klas wewnętrznych.

⁷⁸ Również składowych prywatnych.

⁷⁹ Klasy anonimowe często wykorzystywane są do obsługi zdarzeń w programowaniu aplikacji korzystających z graficznego interfejsu użytkownika.

5. Które z poniższych deklaracji są błędne (KlasaA, KlasaB, KlasaC oznaczają nazwy klas, natomiast InterfejsA, InterfejsB, InterfejsC to nazwy interfejsów)?

```
class KlasaA extends KlasaB
class KlasaA implements InterfejsA
class KlasaC extends KlasaA, KlasaB
class KlasaB implements InterfejsB, InterfejsC
interface InterfejsA implements InterfejsB
interface InterfejsB extends InterfaceA
interface InterfejsC extends InterfaceA, InterfaceB
```

6. Odszukaj w dokumentacji, jakie interfejsy implementuje klasa `String`. Jakie nagłówki metod zdefiniowane zostały w tych interfejsach?
7. Czy poprawny jest poniższy kod?

```
public interface Tester {
}
```

8. Podaj definicję klasy abstrakcyjnej.
9. Wskaż różnice między klasą abstrakcyjną, a interfejsem.
10. Poniższy kod programu zawiera definicję klasy abstrakcyjnej wraz z jej składowymi. Wskaż ewentualne błędy.

```
public abstract class Zwierze {
    private String nazwa;

    public abstract void je() {
        System.out.println ("Aktualnie je.");
    }
    public abstract void idzie();
    public abstract void pije();
}
```

11. Czy możliwe jest utworzenie obiektów na bazie klasy abstrakcyjnej?
12. Czy nazwa klasy, która dziedziczy z klasy abstrakcyjnej i nie dostarcza implementacji metod zawartych w nadklasie musi zostać poprzedzona identyfikatorem `abstract`?
13. Wskaż błąd w poniższym kodzie programu.

```
final class A {
    private String nazwa;

    public String pobierzNazwe() {
        return this.nazwa;
    }
}

class B extends A {
    private int liczba;

    public int pobierzLiczbe() {
        return liczba;
    }
}
```

14. Jakie własności posiadają metody finalne?
15. Czym są klasy anonimowe? Wskaż ich zastosowanie.

Zadania do wykonania

INTERFEJSY I KLASY ABSTRAKCYJNE

Zadanie 118 – *SrodkiTransportu.java*

Napisz program modelujący istniejące środki transportu używane do przemieszczania się zarówno na lądzie, w wodzie, jak i w powietrzu.

Rozwiązanie

Utwórz interfejs `Plywa` oraz `Lata` zawierający metody `plyn()` i `lec()`. Zdefiniuj klasy implementujące każdy z interfejsów (np. `Statek`, `Samolot`) jak i klasę tworzącą przykładowe obiekty wraz z wywołaniem zaimplementowanych metod.

```
interface Plywa {
    public void plyn();
}

interface Lata {
    public void lec();
}

class Statek implements Plywa {
    public void plyn(){
        System.out.println ("Statek płynie");
    }
}

class Samolot implements Lata {
    public void lec(){
        System.out.println ("Samolot leci");
    }
}

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

        Lata samolot = new Samolot();
        samolot.lec();

        Plywa statek = new Statek();
        statek.plyn();

    }
}
```

Zadanie 119 – *Hydroplan.java*

Hydroplan to środek transportu umożliwiający przemieszczanie się zarówno w powietrzu, jak i na wodzie. Zdefiniuj klasę `Hydroplan`, która implementuje obydwa interfejsy (patrz: poprzednie zadanie). Sprawdź poprawność działania programu.

Zadanie 120 – *OdglosyZwierzat.java*

Dokonaj modyfikacji zadania `Zwierzeta.java` występującego we wcześniejszym rozdziale, tworząc interfejs odpowiedzialny za odgłosy zwierząt. Po jego utworzeniu zmodyfikuj klasy, które będą go implementować oraz sprawdź poprawność działania programu. Utwórz trzy nowe klasy reprezentujące 3 kategorie zwierząt. Każda z tych klas powinna implementować pośrednio lub bezpośrednio interfejs `Odglos`.

interface Odglos	nowy interfejs
<code>public void dajGlos()</code>	nagłówek metody (ciało interfejsu)
class Zwierze implements Odglos	klasa implementująca interfejs
<code>private String nazwa</code>	nazwa zwierzęcia
<code>public void dajGlos()</code>	deklaracja metody wymagana przez interfejs

Zadanie 121 – PorządkowanieListyStudentow.java

Dostępna jest lista studentów, którą należy uporządkować według numeru albumu. Napisz program, który zrealizuje opisaną operację.

Rozwiązanie

Utwórz lub wykorzystaj z poprzednich rozdziałów klasę `Student` zawierającą imię, nazwisko oraz numer albumu studenta. Klasa ta powinna również implementować interfejs `Comparable` (zapoznaj się z dokumentacją). Do sortowania tablicy zawierającej studentów użyj metody `sort()` z klasy `java.util.Arrays`. Wyświetl wyniki na konsoli. Poniżej przedstawiona została przykładowa lista studentów

```
Student[] lista = new Student[4];

lista[0] = new Student("Jan", "Kowalski", 432187);
lista[1] = new Student("Adam", "Nowak", 332132);
lista[2] = new Student("Joanna", "Wyszek", 632165);
lista[3] = new Student("Ania", "Nowak", 321419);
```

oraz uzyskane wyniki.

```
Student: Ania Nowak      nr albumu: 321419
Student: Adam Nowak     nr albumu: 332132
Student: Jan Kowalski   nr albumu: 432187
Student: Joanna Wyszek  nr albumu: 632165
```

Interfejs `Comparable` posiada metodę `compareTo(Object)`. Zwraca ona dodatnią, ujemną lub zerową wartość typu `int`, określającą rezultat porównania obiektów. Metoda `sort()` z klasy `java.util.Arrays` umożliwia porządkowanie tablicy składającej się ze zbioru obiektów, wykorzystując wartość zwracaną przez metodę `compareTo()`.

```
import java.util.*;

class Student implements Comparable {

    private String imie;
    private String nazwisko;
    private int nrAlbumu;

    public Student(String imie, String nazwisko, int nrAlbumu) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrAlbumu = nrAlbumu;
    }

    public int compareTo(Object s){
        return (nrAlbumu - ((Student)s).nrAlbumu);
    }

    public String toString() {
        return "Student " + imie + " " + nazwisko + " nr albumu: " + nrAlbumu;
    }
}

public class PorzadkowanieListyStudentow {
    public static void main (String[] args) {
        Student[] lista = new Student[4];

        lista[0] = new Student("Jan", "Kowalski", 432187);
        lista[1] = new Student("Adam", "Nowak", 332132);
        lista[2] = new Student("Joanna", "Wyszek", 632165);
        lista[3] = new Student("Ania", "Nowak", 321419);

        Arrays.sort(lista);

        for(Student student: lista){
            System.out.println(student);
        }
    }
}
```

Zadanie 122 – ZaawansowanePorzadkowanieListyStudentow.java

Zmodyfikuj kod programu porządkującego studentów wg numeru albumu w taki sposób, aby sortowanie odbywało się jednocześnie wg nazwiska, imienia, oraz numeru albumu. Przykładowe rezultaty odnoszące się do danych zawartych w poprzednim zadaniu zostały przedstawione poniżej.

```
Student: Jan Kowalski    nr albumu: 432187
Student: Adam Nowak     nr albumu: 332132
Student: Ania Nowak     nr albumu: 321419
Student: Joanna Wyszek  nr albumu: 632165
```

Zadanie 123 – UrzadzeniaMuzyczne.java

Biuro projektowe firmy BOOMBOX zajmuje się projektowaniem nowych linii elektronicznych urządzeń muzycznych. Przedstaw w języku programowania Java model urządzeń odtwarzających muzykę, wraz z podstawowymi ich właściwościami (włącz, wyłącz, start, stop, pauza), który będzie stanowił podstawę dla dalszych prac projektowych.

Rozwiązanie

Utwórz odpowiednie interfejsy oraz klasę abstrakcyjną dla kilku urządzeń muzycznych. Pierwszy z interfejsów powinien realizować funkcje włączania i wyłączania urządzenia, natomiast drugi funkcje odtwarzające muzykę (start, stop, pauza). Utwórz klasę abstrakcyjną (z wykorzystaniem powyższych interfejsów) modelującą zasadniczą funkcjonalność odtwarzacza oraz klasę reprezentującą dedykowany odtwarzacz CD. Sprawdź poprawność działania programu poprzez utworzenie odtwarzacza CD marki Sony i sprawdzenie jego działania. Poniżej przedstawione zostały deklaracje interfejsów oraz klas koniecznych do utworzenia programu.

interface Wlacz	interfejs dla urządzeń które można włączyć i wyłą.
public void wlacz()	włączenie urządzenia
public void wylacz()	wyłączenie urządzenia
public boolean czyDziala()	sprawdzenie czy urządzenie działa
interface Odtwarza	interfejs dla urządzeń odtwarzających
public void start()	urządzenie odtwarza
public void stop()	urządzenie nie odtwarza
public void pauza()	urządzenie zatrzymane
abstract class Odtwarzacz implements Wlacz, Odtwarza	klasa abstrakcyjna implementująca dwa interfejsy
protected int glosnosc	pole klasy (aktualna głośność)
private boolean dziala	pole klasy (stan urządzenia)
protected String marka	pole klasy (marka urządzenia)
public Odtwarzacz(String)	konstruktor klasy
public void wlacz()	metoda włączająca urządzenie
public void wylacz()	metoda wyłączająca urządzenie
public boolean czyDziala()	metoda dostępową
public abstract void glosniej()	metoda abstrakcyjna
public abstract void ciszej()	metoda abstrakcyjna
class OdtwarzaczCD extends Odtwarzacz	klasa reprezentująca prosty odtwarzacz CD
public OdtwarzaczCD(String)	konstruktor klasy
public void glosniej()	metoda zwiększająca głośność
public void ciszej()	metoda zmniejszająca głośność
public void start()	metoda uruchamiająca odtwarzanie
public void stop()	metoda zatrzymująca odtwarzanie

```

    public void pauza()

    public String toString()

interface Wlacza {
    public void wlacz();
    public void wylacz();
    public boolean czyDziala();
}

interface Odtwarza {
    public void start();
    public void stop();
    public void pauza();
}

abstract class Odtwarzacz implements Wlacza, Odtwarza {
    protected int glosnosc;
    private boolean dziala;
    protected String marka;

    public Odtwarzacz(String marka){
        this.marka = marka;
        dziala = false;
    }

    public void wlacz(){
        dziala = true;
    }

    public void wylacz(){
        dziala = false;
    }
    public boolean czyDziala() {
        return dziala;
    }

    public abstract void glosniej();
    public abstract void ciszej();
}

class OdtwarzaczCD extends Odtwarzacz {

    public OdtwarzaczCD (String marka){
        super(marka);
        glosnosc = 10;
    }

    public void glosniej(){
        glosnosc++;
    }
    public void ciszej(){
        glosnosc--;
    }

    public void start(){
        System.out.println ("Odtwarzacz CD odtwarza");
    }
    public void stop(){
        System.out.println ("Odtwarzacz CD nie odtwarza");
    }
    public void pauza(){
        System.out.println ("Odtwarzacz CD zatrzymany");
    }

    public String toString() {
        return "Odtwarzacz CD marki: " + marka + " jest " +
        ( czyDziala() ? "włączony" : "wyłączony") + " Głośność: " + glosnosc;
    }
}

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

```

metoda wstrzymująca odtwarzanie

metoda zwracająca reprezentację obiektu w formie łańcucha tekstowego

```

        OdtwarzaczCD sony = new OdtwarzaczCD("Sony");

        sony.wlacz();
        sony.start();
        sony.glosniej();

        System.out.println (sony);
    }
}

```

Zadanie 124 – NoweUrządzenieMuzyczne.java

Firma BOOMBOX przygotowuje produkcję kolejnej klasy urządzeń – odtwarzaczy MP3. Zmodyfikuj poprzedni program w taki sposób, aby uwzględnił nową kategorię urządzeń muzycznych. Sprawdź poprawność działania programu poprzez utworzenie osobistego odtwarzacza MP3. Wykorzystaj poniższą deklarację klasy oraz jej metod.

class	OdtwarzaczMP3	klasa reprezentująca prosty odtwarzacz MP3
extends Odtwarzacz		dziedziczka z klasy Odtwarzacz
private int pojemnosc		pole klasy (pojemność urządzenia)
private int bateria		pole klasy (stan baterii)
public OdtwarzaczMP3(String, int, int)		konstruktor klasy
public void glosniej()		metoda zwiększająca głośność
public void ciszej()		metoda zmniejszająca głośność
public void start()		metoda uruchamiająca odtwarzanie
public void stop()		metoda zatrzymująca odtwarzanie
public void pauza()		metoda wstrzymująca odtwarzanie
public String toString()		metoda zwracająca reprezentację obiektu w formie łańcucha tekstowego

Zadanie 125 – Radioodbiornik.java

Wzorując się na poprzednich zadaniach, utwórz model kolejnego urządzenia muzycznego – radioodbiornika. Zastanów się, które interfejsy należy zastosować oraz czy wskazane jest dziedziczenie z klasy abstrakcyjnej. Poniżej zamieszczona została deklaracja klasy wraz z jej składowymi. Sprawdź poprawność działania programu tworząc radioodbiornik.

class Radio		klasa reprezentująca radio implementująca interfejs
implements Wlacz		
private boolean dziala		pole klasy (stan urządzenia)
private String marka		pole klasy (marka urządzenia)
private double czestotliwosc		pole klasy (aktualna częstotliwość)
private int glosnosc		pole klasy (aktualna głośność)
public Radio(String, double)		konstruktor klasy
public void glosniej()		metoda zwiększająca głośność
public void ciszej()		metoda zmniejszająca głośność
public String toString()		metoda zwracająca reprezentację obiektu w formie łańcucha tekstowego

KLASY WEWNĘTRZNE I FINALNE

Zadanie 126 – TelefonKomorkowyZKartaSIM.java

Każdy telefon komórkowy posiada kartę SIM zawierającą informację związane z numerem, operatorem, książką adresową itp. Utwórz model takiego telefonu, korzystając z narzędzi zawartych w języku programowania Java.

Rozwiązanie

Utwórz klasę `TelefonKomorkowyZKartaSIM`, w której zadeklaruj klasę wewnętrzną `KartaSIM`. Następnie utwórz nowy obiekt klasy wewnętrznej oraz pobierz dane z karty SIM. Sprawdź działanie tak utworzonej klasy poprzez utworzenie kilku obiektów. Zwróć uwagę, jakie pliki zostaną wygenerowane podczas kompilacji programu.

```
public class TelefonKomorkowyZKartaSIM {

    private KartaSIM sim;

    public TelefonKomorkowyZKartaSIM(String numer) {
        sim = new KartaSIM(numer);
    }

    // definiujemy klasę wewnętrzną KartaSIM

    class KartaSIM {
        private String numerTelefonu;

        public KartaSIM(String numer) {
            numerTelefonu = numer;
        }

        public String pobierzNumerTelefonu() {
            return numerTelefonu;
        }
    }

    public String toString(){
        return "Telefon z karta SIM o numerze: " + sim.pobierzNumerTelefonu();
    }

    public static void main (String[] args) {
        TelefonKomorkowyZKartaSIM telefonMarcina = new
        TelefonKomorkowyZKartaSIM("507864934");
        TelefonKomorkowyZKartaSIM telefonKasi = new
        TelefonKomorkowyZKartaSIM("679342123");

        System.out.println (telefonMarcina);
        System.out.println (telefonKasi);
    }
}
```

Zadanie 127 – Komputer.java

W skład każdego komputera wchodzi procesor. Posiada on dostęp do poszczególnych elementów składowych komputera (pamięć, grafika, porty itp.). Utwórz klasę `Komputer` zawierającą klasę wewnętrzną `Procesor`. Następnie na jej podstawie utwórz trzy przykładowe komputery z różnymi procesorami. Wykorzystaj poniższe informacje.

```
Komputer dell = new Komputer("Dell", "Pentium IV", 1.66, 1024);
Komputer ibm = new Komputer("IBM", "AMD", 2.2, 2048);
```

class Komputer	klasa główna
<code>private String marka</code>	pole klasy (marka komputera)
<code>private Procesor procesor</code>	pole klasy (rodzaj procesora)
<code>private int pamiec</code>	pole klasy (zainstalowana pamięć w MB)
<code>public Komputer(String, String, double, int)</code>	konstruktor klasy
<code>class Procesor</code>	klasa wewnętrzna (patrz niżej)
<code>public String toString()</code>	metoda zwracająca reprezentację obiektu w formie łańcucha tekstowego
class Procesor	klasa wewnętrzna klasy Komputer
<code>private String marka</code>	pole klasy (marka procesora)

```
private double czestotliwosc  
public Procesor(String, double)  
  
public void rezerwujPamiec(int)  
  
public void zwolnijPamiec(int)  
  
public String toString()
```

pole klasy (częstotliwość procesora w GHz)

konstruktor klasy

metoda rezerwująca określoną w MB liczbę pamięci

metoda zwalniania określoną w MB liczbę pamięci

metoda zwracająca reprezentację obiektu w formie łańcucha tekstowego

Rozdział 8

Strumienie, pliki i obsługa błędów

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- kontrolować prawidłowe działanie programu,
- obsługiwać programowo błędy powstałe w trakcie wykonywania programu,
- definiować obsługę własnych sytuacji wyjątkowych,
- odczytywać oraz zapisywać dane do pliku tekstowego i binarnego,
- zarządzać systemem plików i katalogów systemu operacyjnego,
- przetwarzać strumień danych z sieci Internet,

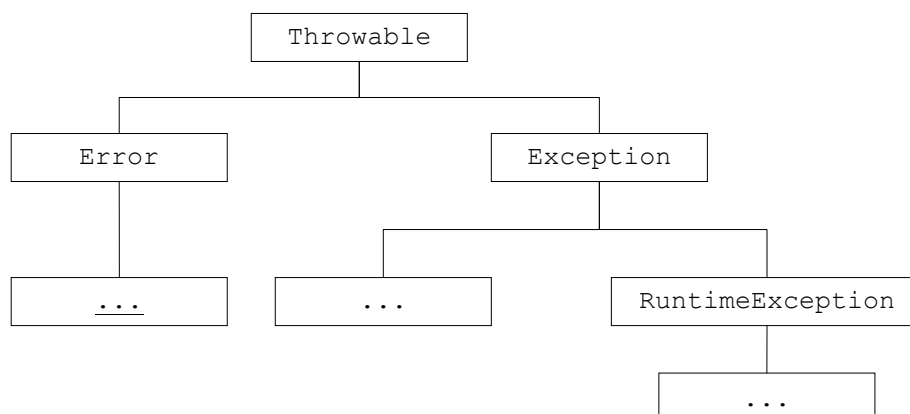
Wprowadzenie do zagadnień

Program komputerowy składa się z ciągu instrukcji wykonywanych przez komputer. Podstawą dla wykonania programu są dane wejściowe, stanowiące źródło informacji. Rezultat działania programu stanowią dane wyjściowe, najczęściej składowane na nośnikach pamięci. Operacje wejścia-wyjścia (ang. *I/O input/output*) stanowią fundament dla komunikacji programu komputerowego z otoczeniem.

Należy zwrócić uwagę, iż operacje mające za zadanie komunikować się z otoczeniem programu mogą stanowić potencjalne źródła błędów. Tworząc kod programu, programista stara się je w maksymalnym stopniu eliminować. Niektóre z nich możliwe są do zdiagnozowania już na etapie kompilacji (dotyczy to przede wszystkim błędów składniowych kodu programu). Jednakże pewne błędy ujawniają się dopiero w momencie wykonania programu. Ich obsługa jest warunkiem dalszego prawidłowego działania programu.

WYJĄTKI I ICH STRUKTURA

Gdy system napotka instrukcje programu, której nie jest w stanie poprawnie wykonać, powstaje sytuacja wyjątkowa. Zgłaszany jest wtedy tzw. wyjątek (ang. *exception*). Jest to obiekt wywodzący się z klasy `Throwable` lub klasy pochodnej:



Rys. 8. Struktura wyjątków.

Poniższy kod zawiera instrukcję, która spowoduje błąd wykonania programu. W przedstawionym przykładzie zostanie zgłoszony wyjątek `ArithmeticException`.

```

public void podziel() {
    System.out.println("Wykonuję operację dzielenia...");
    double x = 5.0/0;    // miejsce powstania wyjątku ArithmeticException
    System.out.println("Rezultat dzielenia: " + x);
    System.out.println("Operacja dzielenia wykonana.");
}

```

Klasa `Exception` oraz jej klasy pochodne opisują te wyjątki, które odnoszą się do wykonywanego programu w Javie. Gdy pojawia się sytuacja wyjątkowa (błąd w programie), tworzony jest obiekt⁸⁰ klasy `Exception` (lub klasy pochodnej) i przekazywany do JVM. Klasa `Error` wraz z jej klasami pochodnymi opisuje te wyjątki, które odnoszą się do JVM. Tworzone na podstawie tych klas obiekty reprezentują poważne błędy systemowe występujące w trakcie wykonywania programu.

WYJĄTKI KONTROLOWANE I NIEKONTROLOWANE

Język Java wprowadza podział wszystkich wyjątków na niekontrolowane oraz kontrolowane. Te ostatnie muszą zostać obsłużone bezpośrednio w kodzie programu. Brak obsługi wyjątku kontrolowanego spowoduje powstanie błędu kompilacji.

Wyjątki kontrolowane (wywodzące się z klasy `Exception`, oprócz klasy `RuntimeException`) pojawiają się najczęściej w sytuacji, gdy wykonywany program próbuje skorzystać z dostępnych zasobów (odczyt lub zapis do pliku, próba dostępu do bazy danych, zasobów w sieci Internet) i operacja taka zakończy się niepowodzeniem. Jeśli wywoływana metoda występująca w kodzie programu zgłasza wyjątek kontrolowany, niezbędne jest jego przechwycenie oraz obsługa za pomocą instrukcji `try..catch` lub też przekazanie go do metody nadrzędnej (wywołującej). Wymagane jest wtedy użycie klauzuli `throws` umieszczonej w nagłówku metody nadrzędnej, która zawierać musi nazwy wszystkich zgłaszanych wyjątków⁸¹:

⁸⁰ Obiekt może zostać utworzony bezpośrednio w programie (instrukcja `throw`) lub też wygenerowany przez JVM.

⁸¹ Brak nazwy wyjątku kontrolowanego w nagłówku metody, w której może on wystąpić powoduje błąd kompilacji.

```

public void odczytZawartosciPliku() throws IOException {
    ...
    // konieczne jest użycie try..catch
    // lub przekazanie wyjątku metodzie nadrzędnej (klauszula throws)
    FileReader fr = new FileReader("dane.txt");
    BufferedReader br = new BufferedReader(fr);
    String line = br.readLine();
    ...
}

```

Przykładami wyjątków kontrolowanych są:

- `FileNotFoundException` (próba dostępu do nieistniejącego pliku),
- `UnknownHostException` (błędny adres dostępu do zasobu w sieci Internet)
- `SQLException` (błąd dostępu do zasobów bazy danych)

Wyjątki niekontrolowane (wywodzące się z klasy `Error` lub `RuntimeException`) związane są najczęściej z błędami logicznymi występującymi w programie lub też błędami JVM. Typowe przykłady to:

- `ArithmeticException` (problem z wykonaniem operacji arytmetycznej, np. próba dzielenia przez zero),
- `ArrayIndexOutOfBoundsException` (odwołanie do nieistniejącego elementu tablicy),
- `OutOfMemoryError` (brak pamięci),
- `StackOverflowError` (przepełnienie pamięci stosu).

Wyjątki wywodzące się z klasy `Error` oznaczają najczęściej poważne problemy systemowe. Stąd ich obsługa nie zawsze jest możliwa.

INSTRUKCJA TRY..CATCH

Jeśli kod programu nie zawiera instrukcji obsługi zgłoszonego wyjątku, program kończy działanie wyświetlając na konsoli informację o powstałym błędzie⁸². Możliwe jest jednak przechwycenie zgłoszonego wyjątku i programowa jego obsługa. Aby możliwe było przechwycenie wyjątku, należy umieścić instrukcje mogące spowodować błąd wewnątrz bloku instrukcji `try`:

```

public void podziel() {
    try {
        System.out.println("Wykonuję operację dzielenia...");
        double x = 5.0/0;
        System.out.println("Rezultat dzielenia: " + x);
        ...
    }
    catch (ArithmeticException e){
        System.out.println("Niestety, dzielenie przez 0 nie jest dozwolone!");
        System.out.println("Wprowadź inne, poprawne argumenty działania.");
    }
    finally {
        // dodatkowe instrukcje (opcjonalnie)
    }
    // dalsze instrukcje metody
    System.out.println("Operacja dzielenia wykonana.");
    ...
}

```

Jeśli którakolwiek z instrukcji umieszczonych wewnątrz bloku `try` spowoduje błąd, wykonanie programu zostanie przerwane, a sterowanie przekazane do bloku instrukcji `catch`⁸³. Jeśli typ powstałego wyjątku jest zgodny z wymienionym typem w instrukcji `catch`, zostaną wykonane wszystkie instrukcje znajdujące się wewnątrz bloku `catch`. Następnie sterowanie przebiegiem wykonania programu zostanie przekazane do kolejnych instrukcji występujących po bloku `catch`. Możliwe jest umieszczenie po `catch` bloku `finally`, którego instrukcje zostaną wykonane niezależnie od powstałych w bloku `try` błędów. W bloku `finally` umieszczane są z reguły polecenia, zadaniem

⁸² Wyświetlany komunikat zawiera nazwę wyjątku oraz miejsce, w którym został zgłoszony.

⁸³ Blok programu `try` dopuszcza występowanie dowolnej liczby bloków `catch()`. Kolejność ich występowania jest istotna i ściśle zależna od struktury klas (zobacz schemat struktury wyjątków).

których jest zarządzanie zasobami systemu (np. zamknięcie pliku, zwolnienie nieużywanego zasobu pamięci).

Jeśli wyjątek nie zostanie przechwycony i obsługony w metodzie, w której wystąpił, przekazywany jest do metod wywołujących, z których ostatnia to metoda `main()`. Gdy w żadnej z metod nie nastąpi przechwycenie i obsługa wyjątku, program kończy działanie wyświetlając na konsoli komunikat o powstałym błędzie.

WYRZUCANIE WYJĄTKU

Oprócz wyjątków zgłaszanych przez JVM, możliwe jest programowe zgłaszanie wyjątków bezpośrednio w kodzie programu. Poniższy fragment zawiera instrukcję zgłaszającą wyjątek (`throw`) w przypadku, gdy nastąpi próba odwołania do nieistniejącego elementu tablicy:

```
int[] tab = new int[3];
for (int i=0; i<=3; i++) {
    if (i >= 3) {
        throw new ArrayIndexOutOfBoundsException("Przekroczony zakres tablicy");
    }
    tab[i] = 5;
}
```

TWORZENIE NOWEJ KLASY WYJĄTKU

Mechanizm obsługi wyjątków dostępny w języku Java umożliwia rozbudowę struktury klas o nowe kategorie wyjątków. Nowe klasy należy tworzyć na podstawie klasy `Exception` lub klas pochodnych. Klasy takie zawierają najczęściej zadeklarowane jawnie konstruktory umożliwiające przekazanie informacji o powstałym błędzie.

```
class WyjatekLiczbaNaturalna extends Exception {
    public WyjatekLiczbaNaturalna() {
        super("Nieprawidłowa liczba naturalna.");
    }
    public WyjatekLiczbaNaturalna(String komunikat) {
        super(komunikat);
    }
}
```

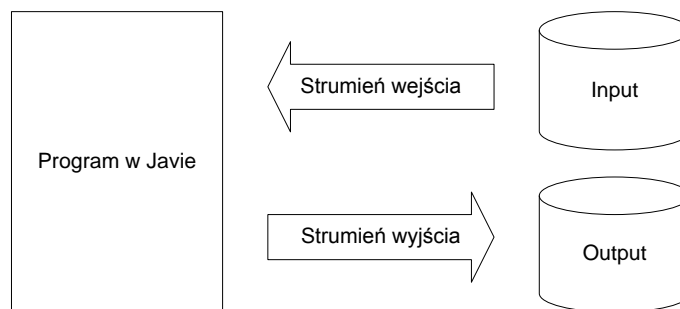
Po utworzeniu nowej klasy wyjątku możliwe jest zgłaszanie wyjątków utworzonego typu.

```
if (liczba < 1) {
    throw new WyjatekLiczbaNaturalna();
}
```

Zgłoszony wyjątek może być następnie przechwycony i obsługiwany za pomocą instrukcji `try..catch`.

STRUMIENIE DANYCH

Operacje wejścia-wyjścia w języku Java opierają się na pojęciu „strumienia” (ang. *stream*), rozumianego jako kanał transmisyjny, przez który przepływa sekwencja bajtów ze źródła do programu lub z programu do miejsca przeznaczenia. Gdy dane przekazywane są ze źródła do programu, mowa jest o strumieniu wejściowym (ang. *input stream*), natomiast jeśli dane przekazywane są z programu do miejsca przeznaczenia, wtedy określane jest to jako strumień wyjściowy (ang. *output stream*). Źródłem, czy miejscem przeznaczenia danych mogą być pliki dyskowe, urządzenia (np. drukarka, skaner), inne programy, czy zasoby danych dostępne w sieci (np. zasoby sieci Internet).



Rys. 9. Wejściowe i wyjściowe strumienie danych.

Istnieje pokaźna grupa klas bibliotecznych do obsługi strumieni danych. Ponieważ Java dzieli strumienie danych na strumienie wejścia oraz wyjścia, stąd też zbiór klas bibliotecznych został również podzielony wg tych dwóch kategorii. Poniższa tabela przedstawia klasy bazowe.

Tabela 3. Klasy bazowe dla operacji wejścia-wyjścia.

STRUMIEŃ	ZNAKOWY	BINARNY
Wejściowy (input)	java.io.Reader	java.io.InputStream
Wyjściowy (output)	java.io.Writer	java.io.OutputStream

W zależności od charakteru źródła lub miejsca przeznaczenia danych wykorzystywane są dedykowane klasy pochodne.

STRUMIENIE PLIKOWE

Sekwencyjny dostęp do plików⁸⁴ realizowany jest poprzez klasy `FileInputStream`, `FileOutputStream` oraz `FileReader`, `FileWriter`. Dwie pierwsze umożliwiają dostęp (odczyt oraz zapis) do danych binarnych (pliki graficzne, dźwiękowe,...), a także do danych tekstowych w formacie ASCII, natomiast dwie ostatnie realizują dostęp do plików tekstowych w formacie Unicode. Poniższy program ilustruje odczyt zawartości pliku tekstowego. Należy zwrócić uwagę na konieczność użycia obsługi błędów.

```
import java.io.*;

public class OdczytPlikuTekstowego {
    public static void main(String[] args) {
        FileReader plik = null;
        int znak;
        try {
            // obiekt reprezentujący wskazany plik tekstowy
            plik = new FileReader("dane.txt");
            // odczyt kolejnych znaków z pliku
            while ( (znak = plik.read()) != -1 ) {
                // wyświetlenie odczytanego znaku na konsoli
                System.out.print((char) znak);
            }
        }
        catch (FileNotFoundException e) {
            System.out.println("Brak pliku o podanej nazwie!");
        }
        catch (IOException e) {
            System.out.println("Problem z odczytem pliku!");
        }
        finally {
            if (plik != null)
                try {
                    plik.close();
                }
        }
    }
}
```

⁸⁴ Wyróżnić można sekwencyjny dostęp do plików (ang. *sequential-access*) oraz dostęp swobodny (ang. *random-access*).

```

        catch (IOException e){
            System.out.println("Problem z zamknięciem pliku!");
        }
    }
}

```

Dla zwiększenia wydajności operacji wejścia–wyjścia odczyt oraz zapis danych może być buforowany, co realizowane jest poprzez użycie klas `BufferedInputStream`, `BufferedOutputStream`, `BufferedReader`, `BufferedWriter`.

```

FileReader plik = new FileReader("dane.txt");
BufferedReader plikBuforowany = new BufferedReader(plik);
...
String liniaTekstu;
liniaTekstu = plikBuforowany.readLine();
...

```

Dodatkowo klasa `BufferedReader` posiada metodę `readLine()`, umożliwiającą odczyt pojedynczej linii z pliku tekstowego.

STRUMIENIE SIECIOWE

W dobie Internetu trudno wyobrazić sobie język programowania, który nie umożliwiałby dostępu oraz korzystania z zasobów zgromadzonych w sieci. Język programowania Java, dzięki rozbudowanej bibliotece klas, posiada silne wsparcie dla wykonywania takich operacji. Jednym z podstawowych narzędzi umożliwiających dostęp do zasobów sieciowych jest klasa `java.net.URL`. Dzięki zastosowaniu ujednoliconego formatu adresowania URL (ang. *Uniform Resource Locator*), możliwe jest wykorzystanie dowolnego zasobu udostępnionego w sieci WWW. Może nim być zarówno plik, katalog, ale także, poprzez wykorzystanie dostępnych protokołów, dostęp do zasobów zgromadzonych w bazach danych.

PLIKI I KATALOGI

Obiekt klasy `File` reprezentuje pojedynczy plik w systemie lub katalog (również nieistniejący). Szereg metod tej klasy pozwala na operowanie na plikach lub katalogach systemu. Poniższy przykład ilustruje użycie klasy `File` do wyświetlenia zawartości katalogu Windows:

```

import java.io.*;

public class ZawartoscFolderu {
    public static void main(String[] args) {
        String nazwaFolderu = "C:" + File.separator + "WINDOWS";
        File folder = new File(nazwaFolderu);
        for(String nazwaPlikuLubFolderu : folder.list())
            System.out.println(nazwaPlikuLubFolderu);
    }
}

```

Ponadto metody klasy `File` umożliwiają w szczególności utworzenie nowego pliku, usuwanie plików, tworzenie folderów, zmianę atrybutów plików, zmianę nazwy pliku, sprawdzenie statusu (plik, czy katalog). Pełny wykaz metod dostępny jest w Java API.

Pytania sprawdzające

1. Wymień zalety stosowania procedury obsługi wyjątków.
2. Co oznaczają następujące wyjątki: `ArrayIndexOutOfBoundsException`, `NumberFormatException`, `NullPointerException`?
3. Jaki wyjątek zgłasza metoda `charAt()` klasy `String`? Sprawdź w dokumentacji Javy.
4. Czym charakteryzują się wyjątki kontrolowane? Z jakich klas się wywodzą?
5. Do jakiej kategorii należy wyjątek `FileNotFoundException`?
6. Scharakteryzuj składnię instrukcji `try...catch`.

7. Wskaż zastosowania, jakie pełni blok instrukcji `finalny`.
8. Jaką funkcję pełni klauzula `throws` w nagłówku metody?
9. Podaj nazwy klas, z których dziedziczyć mogą nowo tworzone wyjątki.
10. Omów składnię instrukcji `throw`. Jaką funkcję pełni ta instrukcja? Podaj przykłady użycia.
11. Wskaż różnice pomiędzy klasami `Reader` oraz `InputStream`.
12. Język Java zawiera parę klas: `InputStreamReader` oraz `OutputStreamWriter`. Jaki jest cel stosowania tych klas?
13. Jaką funkcję pełni metoda `mkdirs()` klasy `java.io.File`?
14. Wymień metody umożliwiające uzyskanie szczegółowych informacji o dowolnym pliku dostępnym na nośniku pamięci.
15. Jaką funkcję pełni pole statyczne `File.separator`? Podaj cel jego stosowania w kontekście tworzenia programów uruchamianych w otoczeniu różnych systemów operacyjnych.
16. W jaki sposób można uzyskać znak końca linii, niezależny od stosowanego systemu operacyjnego?
17. Co oznacza skrót URL? Podaj składnię jego zapisu.
18. W jaki sposób realizowane jest dopisywanie danych na końcu pliku? Co stanie się, gdy plik do zapisu zostanie otwarty z wartością parametru `append` równą `false`?
19. Operacja buforowania dostępu do danych pozwala na zwiększenie wydajności odczytu oraz zapisu informacji. W jaki sposób jest realizowana?
20. Co oznacza wyjątek: `MalformedURLException`? W jakiej sytuacji jest generowany?

Zadania do wykonania

OBSŁUGA BŁĘDÓW

Zadanie 128 – *OceanWspak.java*

Poniższy kod zawiera program wyświetlający na konsoli napis `wspak`. Dokonaj analizy programu i w miejsce **TYP-WYJĄTKU** wprowadź poprawną nazwę klasy. W przypadku trudności sprawdź w dokumentacji Javy, jaki wyjątek zgłaszany jest przez metodę `charAt()` klasy `String`.

```
public class OceanWspak {

    public static void main(String[] args) {
        String ocean = "Ocean Indyjski";
        try {
            for(int i=ocean.length(); i>=0; i--)
                System.out.print(ocean.charAt(i));
        }
        catch(TYP-WYJĄTKU e){
            System.out.println("Program został wykonany nieprawidłowo!");
        }
    }
}
```

Zadanie 129 – *SumaLiczbaCalkowitych.java*

Napisz program, którego zadaniem jest wyznaczenie sumy liczb całkowitych wprowadzonych z konsoli.

Rozwiązanie

Poniższy kod programu zawiera rozwiązanie zadania. Zwróć uwagę na zawartą w kodzie programu obsługę błędów umożliwiającą weryfikację poprawności wprowadzonych danych.

```
import java.util.*;

public class SumaLiczbaCalkowitych {

    public static void main(String[] args) {
        int sumaLiczba = 0;
        int liczbaCalkowita = 0;
        System.out.println("Wprowadź liczby całkowite.");
        System.out.println("Wartość 0 kończy wprowadzanie.\n");
        Scanner sc = new Scanner(System.in);

        do {
            try {
                System.out.print("Liczba całkowita: ");
                liczbaCalkowita = sc.nextInt();
                sumaLiczba += liczbaCalkowita;
            }
            catch (InputMismatchException e) {
                System.out.println("Wprowadzona wartość \""
                    + sc.nextLine() + "\" jest niepoprawna!!!");
            }
        } while (liczbaCalkowita!=0);

        System.out.println("Suma wprowadzonych wartości: " + sumaLiczba);
    }
}
```

Zadanie 130 – DzieleniePrzezZero.java

Napisz program obliczający iloraz dwóch liczb całkowitych wprowadzonych z konsoli.

Rozwiązanie

Uwzględnij w programie możliwość wystąpienia sytuacji wyjątkowych:

- użytkownik wprowadzi nieprawidłowe dane (np. ciąg znaków nie będący liczbą całkowitą),
- wartość dzielnika wynosi zero.

Zadanie 131 – WielkoscWynagrodzenia.java

Grzegorz Wojdacki jest pracownikiem fizycznym firmy MILENA sp. z o.o. Jego wynagrodzenie wyznaczone jest wg formuły:

wynagrodzenie = liczba przepracowanych godzin x stawka za 1 godzinę w zł i gr.

Napisz program, który na podstawie tych dwóch wartości (liczba godzin, stawka) odczytanych z wiersza poleceń obliczy wielkość wynagrodzenia pracownika. Uwzględnij w programie możliwość wystąpienia sytuacji wyjątkowych.

Zadanie 132 – PierwiastekKwadratowy.java

Poniższy kod programu zawiera metodę wyznaczającą wartość pierwiastka kwadratowego dla dowolnej nieujemnej wartości rzeczywistej:

```
public static double pierwiastekKwadratowy(double x) {
    if(x<0) {
        throw new IllegalArgumentException("Brak pierwiastków dla wartości ujemnych");
    } else {
        return Math.sqrt(x);
    }
}
```

Napisz program, który stosując powyższą metodę obliczy wartość pierwiastka kwadratowego dla dowolnej wartości rzeczywistej odczytanej z konsoli. Uwzględnij w programie możliwość wystąpienia sytuacji wyjątkowych (użyj instrukcji try..catch). Następnie spróbuj wykonać obliczenia dla niedopuszczalnej wartości, np. -7.

Zadanie 133 – SredniaPredkoscRoweru.java

Licznik rowerowy podaje całkowity przebyty dystans oraz efektywny czas jazdy roweru (bez postojów). Utwórz metodę o nagłówku jak poniżej, która na podstawie tych dwóch wartości obliczy prędkość jazdy roweru zgodnie ze wzorem $v=s/t$, gdzie s oznacza przebyty dystans, a t czas jazdy.


```
public static double sredniaPredkoscRoweru(double s, double t) {}
```

Następnie napisz program, który na podstawie danych odczytanych z konsoli wyznaczy średnią prędkość jazdy roweru.

Rozwiązanie

Utworzona metoda powinna weryfikować poprawność wartości przekazywanych parametrów generując wyjątek, gdy wartości te są niedopuszczalne.

Zadanie 134 – SystemyLiczbowe.java

System szesnastkowy to system o podstawie 16, wykorzystujący do zapisu 16 znaków (cyfry 0..9 oraz litery A, B, C, D, E i F). Poniższy nagłówek określa metodę, zadaniem której jest konwersja liczby z systemu szesnastkowego na dziesiętny:

```
public static long hex2dec(String hex) {}
```

Utwórz powyższą metodę uwzględniając następujące założenie: metoda powinna zgłaszać wyjątek, jeśli wartość parametru (hex) nie jest prawidłową liczbą w systemie szesnastkowym. Następnie napisz program, który na podstawie wartości parametru wiersza poleceń dokona konwersji odczytanej wartości w systemie szesnastkowym na system dziesiętny.

Zadanie 135 – TemperaturaPomieszczenia.java

W pomieszczeniu zamontowano regulator temperatury sterowany komputerowo. Poniższy program napisany w języku Java steruje urządzeniem. Uzupełnij kod programu we wskazanym miejscu, aby zawierał on ustawienie następujących wartości temperatury: 19.6C, 17.2C, 27.4C. Po każdej zmianie wartości temperatury wyświetl jej bieżącą wartość na konsoli.

```
public class TemperaturaPomieszczenia {
    private final double TEMP_MAX = 25.0;
    private final double TEMP_MIN = 15.0;
    private double temperatura;

    public static void main(String[] args) {
        TemperaturaPomieszczenia kuchnia = new TemperaturaPomieszczenia();
        System.out.println("ELEKTRONICZNY REGULATOR TEMPERATURY");
        // tu zmień i odczytaj temp.
    }

    public void ustawTemperature(double temp) throws TemperaturaException {
        if ((temp < TEMP_MIN) || (temp > TEMP_MAX)) {
            throw new TemperaturaException();
        } else {
            temperatura = temp;
        }
    }

    public double pobierzTemperature() {
        return temperatura;
    }
}

class TemperaturaException extends Exception {
    public TemperaturaException() {
        super("Temperatura poza dopuszczalnym zakresem.");
    }
    public TemperaturaException(String komunikat) {
        super(komunikat);
    }
}
```

PLIKI I KATALOGI

Zadanie 136 – PojemnoscDyskow.java

Większość komputerów wyposażona jest w pamięć masową. Napisz program wyświetlający na konsoli wykaz dostępnych nośników pamięci wraz z informacją o ich całkowitej pojemności.

Rozwiązanie

Zapoznaj się z wykazem dostępnych metod klasy `java.io.File`.

```
import java.io.*;

public class PojemnoscDyskow {

    public static void main(String[] args) {
        String folderGlowny = File.separator;
        String zasob;
        double pojemnosc;
        double GB = 1024*1024*1024;
        System.out.println("NAZWA POJEMNOŚĆ");

        File[] dyski = (new File(folderGlowny)).listRoots();
        for(File dysk : dyski) {
            zasob = dysk.getPath();
            pojemnosc = dysk.getTotalSpace()/GB;
            System.out.printf("%4s %8.2fGB\n", zasob, pojemnosc);
        }
    }
}
```

Zadanie 137 – ParametryDyskow.java

Napisz program, który dla każdego z nośników pamięci dostępnych w systemie wyświetli na konsoli informacje o jego całkowitej pojemności, ilości przestrzeni zajętej oraz wolnej. Informacje wyświetl w MB.

Zadanie 138 – FolderDyskuSystemowego.java

Napisz program wyświetlający zawartość folderu głównego dysku systemowego.

Rozwiązanie

```
import java.io.*;

public class FolderDyskuSystemowego {
    public static void main(String[] args) {
        String nazwaFolderu = "C:" + File.separator;
        File folder = new File(nazwaFolderu);
        for(String nazwaPlikuLubFolderu : folder.list())
            System.out.println(nazwaPlikuLubFolderu);
    }
}
```

Zadanie 139 – ZawartoscFolderu.java

Napisz program wyświetlający na konsoli zawartość dowolnego folderu wskazanego w wierszu poleceń. W przypadku plików wyświetl ich nazwę, rozmiar oraz datę ostatniej modyfikacji.

Rozwiązanie

Zapoznaj się z metodami klasy `java.io.File`: `listFiles()`, `isFile()`, `isDirectory()`, `length()`, `lastModified()`

Zadanie 140 – ObjetoscPlikowFolderu.java

Napisz program obliczający całkowitą objętość plików zawartych we wskazanym katalogu. Przykładowe wywołanie programu:

```
java ObjetoscPlikowFolderu C:\WINDOWS
```

Zadanie 141 – StrukturaFolderow.java

Folder (inaczej katalog), związany z logiczną organizacją danych na nośnikach pamięci, umożliwia uporządkowanie informacji zapisanych na dysku. Napisz program, który w katalogu bieżącym utworzy poniższą strukturę folderów.

```

KOMPUTER
DESKTOP
LAPTOP
IBM
COMPAQ
PALMTOP

```

Zadanie 142 – *TworzenieFolderow.java*

Napisz program, który we wskazanym katalogu utworzy folder o podanej nazwie. Informację, gdzie utworzyć nowy folder oraz nazwę folderu do utworzenia odczytaj z wiersza poleceń.

STRUMIENIE PLIKOWE

Dla wykonania zadań zawartych w tym punkcie utwórz plik tekstowy o nazwie `RedutaOrdon.txt` zawierający fragment utworu Adama Mickiewicza.

```

Nam strzelać nie kazano. Wstąpiłem na działo
I spojrziałem na pole; dwieście armat grzmiało.
Artyleryi ruskiej ciągną się szeregi,
Prosto, długo, daleko, jako morza brzegi;
I widziałem ich wodza: przybiegł, mieczem skinął
I jak ptak jedno skrzydło wojska swego zwinął;
Wylewa się spod skrzydła ściśniona piechota
Długą czarną kolumną, jako lawa błota,
Nasypana iskrami bagnetów. Jak sępy
Czarne chorągwie na śmierć prowadzą zastępy.

```

```

Przeciw nim sterczy biała, wąska, zastrzona,
Jak głaz bodzący morze, reduta Ordon.
Sześć tylko miała armat; wciąż dymią i świecą;
I nie tyle prędkich słów gniewne usta miecą,
Nie tyle przejdzie uczuć przez duszę w rozpacz,
Ile z tych dział leciało bomb, kul i kartaczy.
Patrz, tam granat w sam środek kolumny się nurza,
Jak w fale bryła lawy, pułk dymem zachmurza;
Pęka wśród dymu granat, szyk pod niebo leci
I ogromna łysina wśród kolumny świeci.

```

```

Tam kula, lecąc, z dala grozi, szumi, wyje.
Ryczy jak byk przed bitwą, miota się, grunt ryje;
Już dopadła; jak boa wśród kolumn się zwija,
Pali piersią, rwie zębem, oddechem zabija.
Najstraszniejszej nie widać, lecz słyszać po dźwięku,
Po waleniu się trupów, po ranionych jęku:
Gdy kolumnę od końca do końca przewierci,
Jak gdyby środkiem wojska przeszedł anioł śmierci.

```

Zadanie 143 – *RedutaOrdon.java*

Plik tekstowy `RedutaOrdon.txt` zawiera fragment utworu Adama Mickiewicza. Napisz program wyświetlający treść utworu na konsoli.

Rozwiązanie

Wykorzystaj przykładowy kod programu zawarty w części teoretycznej.

Zadanie 144 – *NumerowanieWierszy.java*

Napisz program, wyświetlający na konsoli ponumerowane wiersze pliku tekstowego `RedutaOrdon.txt` zgodnie z poniższym przykładem:

```

01. Nam strzelać nie kazano. -- Wstąpiłem na działo
02. I spojrziałem na pole; dwieście armat grzmiało.
03. Artyleryi ruskiej ciągną się szeregi,
...

```

Zadanie 145 – ZawartoscPlikuTekstowego.java

Napisz program wyświetlający na konsoli zawartość dowolnego pliku tekstowego o nazwie podanej w wierszu polecenia.

Zadanie 146 – BuforowanieDanych.java

Napisz program, który realizował będzie buforowany odczyt zawartości pliku tekstowego. Korzystając z programu, wyświetl na konsoli zawartość pliku `RedutaOrdona.txt`.

Zadanie 147 – StatystykaZnakowPliku.java

Napisz program obliczający liczbę samogłosek oraz spółgłosek występujących w pliku `RedutaOrdona.txt`. Uzyskane rezultaty wyświetl na konsoli.

Zadanie 148 – StatystykaPliku.java

Praktycznie każdy edytor tekstu posiada możliwość określenia liczby znaków, czy wyrazów występujących w edytowanym dokumencie. Napisz program, który dla pliku `RedutaOrdona.txt` wyznaczy liczbę znaków, wyrazów oraz wierszy występujących w utworze. Uzyskane rezultaty wyświetl na konsoli.

Zadanie 149 – KrainyGeograficzne.java

Napisz program, który na podstawie danych odczytanych z wiersza poleceń umożliwia utworzenie pliku tekstowego z wykazem krain geograficznych.

Rozwiązanie

Utwórz program na podstawie poniższego kodu. Korzystając z programu wprowadź do pliku tekstowego następujące nazwy krain geograficznych: Karpaty, Wyżyna Małopolska, Pojezierze Mazurskie, Nizina Wielkopolska, Wyżyna Krakowsko-Częstochowska.

```

import java.io.*;

public class KrainyGeograficzne {

    public static void main(String[] args) {
        FileWriter plik = null;
        final boolean dopisywanie = true; // tryb zapisu do pliku
        String EOL = System.getProperty("line.separator"); // znak końca linii

        try {
            // obiekt reprezentujący wskazany plik tekstowy
            plik = new FileWriter("KrainyGeograficzne.txt", dopisywanie);
            plik.write(args[0] + EOL);
            System.out.println("Zapisano do pliku: " + args[0]);
        } catch (IOException e) {
            System.out.println("Problem z zapisem do pliku!");
        } finally {
            if (plik != null)
                try {
                    plik.close();
                } catch (IOException e) {
                    System.out.println("Problem z zamknięciem pliku!");
                }
        }
    }
}

```

Zadanie 150 – DaneStudentaCSV.java

CSV (ang. *Comma Separated Values*) to format przechowywania danych w plikach tekstowych, gdzie poszczególne informacje rozdzielone są znakiem przecinka. Napisz program, który umożliwia odczytanie z konsoli danych personalnych studenta (nazwisko, imię, wiek, kod pocztowy, miejscowość) i dopisanie ich do pliku `DaneStudenta.txt`. Struktura pliku tekstowego została przedstawiona poniżej:

Badura, Monika, 23, 31-500, Kraków
 Wójcicki, Marek, 22, 00-986, Warszawa
 Maj, Robert, 23, 30-824, Kraków

Wypełnij plik `DaneStudenta.txt` danymi (min. 5 pozycji). Następnie spróbuj otworzyć go w aplikacji arkusz kalkulacyjny Excel, obsługującym ten format danych.

Zadanie 151 – RedutaOrdonHTML.java

Odszukaj w sieci Internet opis struktury dokumentu HTML. Następnie napisz program, który utworzy plik `RedutaOrdon.html` zawierający zawartość pliku `RedutaOrdon.txt`. Poszczególne wiersze utworu wyświetl kursywą. Otwórz tak utworzony plik w dowolnej przeglądarce internetowej.

Zadanie 152 – KonwersjaZnakowPliku.java

Napisz program konwertujący we wskazanym pliku tekstowym litery małe na wielkie, a wielkie na małe. Nazwę pliku oraz jego lokalizację odczytaj z konsoli. Zastosuj program do konwersji znaków w pliku `RedutaOrdon.txt`.

Zadanie 153 – SzyfrCezara.java

Odszukaj w Internecie informacje dotyczące szyfru Cezara. Następnie napisz program szyfrujący dowolny plik tekstowy. Przykładowe wywołanie programu:

```
java SzyfrCezara C:\pliki\RedutaOrdon.txt D:\wyniki\RedutaOrdonZaszyfrowana.txt
```

STRUMIENIE SIECIOWE

Zadanie 154 – ZasobySieciInternet.java

Napisz program wyświetlający na konsoli zawartość wskazanej strony WWW.

Rozwiązanie

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

public class ZasobyInternetu {

    public static void main(String[] args) {
        String adres = "http://www.uek.krakow.pl";
        try
        {
            URL url = new URL(adres);
            InputStream inStream = url.openStream();

            BufferedReader dane =
                new BufferedReader(new InputStreamReader(inStream));

            String linia = "";

            while ((linia = dane.readLine()) != null)
                System.out.println(linia);
        }
        catch (Exception e)
        {
            System.out.println(e.toString());
        }
    }
}
```

Zadanie 155 – KursyWalutNBP.java

Strona internetowa <http://www.nbp.pl/Kursy/KursyA.html> zawiera tabelę średnich kursów walut obcych. Napisz program, który na podstawie danych zawartych w tabeli wyświetli na konsoli średni kurs następujących walut: euro, dolar amerykański, funt szterling, frank szwajcarski.

Rozdział 9

Aplety

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- tworzyć aplikacje uruchamiane w środowisku przeglądarki internetowej,
- przekazywać dane pomiędzy dokumentem HTML, a apletem,
- zmieniać dynamicznie zawartość apletu,
- uwzględniać ograniczenia wynikające ze stosowania apletów.

Wprowadzenie do zagadnień

Aplet stanowi aplikację graficzną utworzoną w języku programowania Java, uruchamianą w środowisku przeglądarki internetowej. Strona WWW tworzona jest zwykle przy użyciu języka HTML (XHTML) i posiada następującą strukturę:

```
<html>
  <head>
    <title>tytuł dokumentu</title>
  </head>
  <body>

    treść dokumentu ...

    <applet
      codebase = codebaseURL (lokalizacja plików klas)
      archive  = archiveList (archiwum plików JAR)
      code     = appletFile (nazwa pliku klasy)
      alt      = alternateText (tekst alternatywny)
      name     = appletInstanceName (nazwa instancji apletu)
      width    = pixels (wysokość apletu w pikselach)
      height   = pixels (szerokość apletu w pikselach)
      align    = alignment (wyrównanie apletu)
      vspace   = pixels HSPACE = pixels (dodatkowy margines wokół apletu)
    >
      <param name = appletAttribute1 VALUE = value>
      <param name = appletAttribute2 VALUE = value>
      . . .
      alternatywny HTML
    </applet>

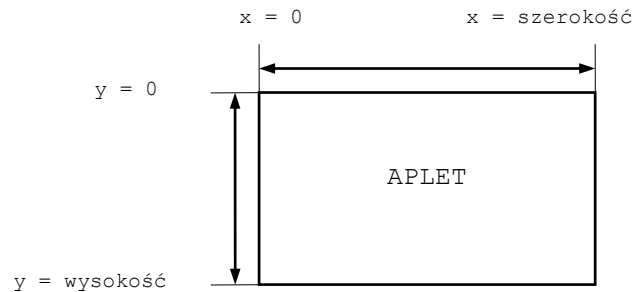
    treść dokumentu ...

  </body>
</html>
```

Podstawowe właściwości dotyczące apletu zawarte są w elemencie APPLET, umieszczonym w treści dokumentu HTML. Wartością atrybutu CODE jest nazwa pliku (.class) zawierającego kod programu do wykonania. Atrybuty WIDTH i HEIGHT określają szerokość oraz wysokość apletu w pikselach⁸⁵. Na

⁸⁵ Wymagane jest użycie jedynie atrybutów CODE, WIDTH oraz HEIGHT. Zastosowanie pozostałych atrybutów jest opcjonalne.

szczególną uwagę zasługują elementy `PARAM` umożliwiające przekazywanie wartości z dokumentu HTML bezpośrednio do apletu.



Rys. 10. Określanie gabarytów apletu.

Tworzenie programu uruchamianego w środowisku przeglądarki internetowej polega na utworzeniu nowej klasy, dziedziczącej z klasy `java.applet.Applet`⁸⁶ i przesłaniającej 4 podstawowe metody⁸⁷ wchodzące w skład struktury apletu.

```
public class MojAplet extends Applet {
    . . .
    public void init() { . . . }
    public void start() { . . . }
    public void stop() { . . . }
    public void destroy() { . . . }
    . . .
}
```

Zadaniem wymienionych metod jest reakcja na zdarzenia związane z obsługą apletu⁸⁸. Metoda:

- `init` – wywoływana zostaje podczas ładowania apletu,
- `start` – wywoływana jest automatycznie po załadowaniu apletu (po metodzie `init()`) oraz za każdym razem, gdy użytkownik powróci na stronę WWW, która zawiera aplet,
- `stop` – wykonywana zostaje każdorazowo, gdy użytkownik opuszcza stronę WWW zawierającą aplet,
- `destroy` – przeznaczona jest do zwalniania zasobów systemu w momencie zamykania przeglądarki internetowej.

Otwarcie strony WWW w przeglądarce internetowej zawierającej aplet powoduje załadowanie pliku apletu (plik `.class`). Następnie tworzona jest instancja pochodnej klasy `java.awt.Applet`, zawierająca wymienione metody. Po zakończeniu inicjalizacji apletu następuje jego uruchomienie. W celu uzyskania graficznej reprezentacji działania apletu wewnątrz przeglądarki internetowej należy użyć metody `paint()`.

Uruchomienie aplikacji utworzonej w języku Java w środowisku przeglądarki internetowej wiąże się z wykonaniem szeregu czynności. Najistotniejsze z nich to:

- utworzenie kodu programu (plik / pliki źródłowe `*.java`),
- kompilacja utworzonych plików źródłowych do postaci kodu bajtowego (pliki `*.class`),
- utworzenie dokumentu HTML, zawierającego wywołanie aplikacji w języku Java (plik `*.class`),
- otwarcie przygotowanego dokumentu HTML w przeglądarce internetowej.

Poniższy przykład prezentuje zestaw plików w języku Java oraz HTML, niezbędnych do wyświetlenia w oknie przeglądarki internetowej apletu zawierającego dane kontaktowe firmy. Aby to uzyskać należy:

⁸⁶ Aplety mogą dziedziczyć po klasie `javax.swing.JApplet` (jeśli wykorzystywany jest w apletach graficzny interfejs użytkownika GUI - Swing) lub `java.applet.Applet` (jeśli stosowane są własne procedury rysujące przy wykorzystaniu dostępnych metod klasy `Graphics`).

⁸⁷ Nie wszystkie wymienione metody muszą zostać przesłonięte.

⁸⁸ Zobacz: <http://java.sun.com/docs/books/tutorial/deployment/applet/appletMethods.html>

- utworzyć plik `DaneKontaktowe.java` wprowadzając podaną poniżej zawartość, a następnie dokonać jego kompilacji,
- utworzyć plik `WitrynaFirmy.html` wprowadzając podaną poniżej zawartość,
- otworzyć plik `WitrynaFirmy.html` w przeglądarce internetowej.

```
import java.applet.*;    // klasy do tworzenia apletów
import java.awt.*;       // klasy do projektowania aplikacji
                        // uruchamianych w środowisku graficznym

public class DaneKontaktowe extends Applet {
    // metoda wywoływana automatycznie, gdy aplet ma zostać uruchomiony
    public void init(){
    }

    // metoda wywoływana, gdy działanie apletu zostaje wstrzymane, np.
    // w sytuacji, gdy użytkownik wyświetli kolejną stronę internetową
    // lub zamknie okno przeglądarki
    public void stop(){
    }

    // metoda umożliwiająca użytkownikowi rysowanie elementów graficznych
    // w obszarze apletu
    public void paint(Graphics g){
        //rysuj tekst
        g.drawString("Pracownia graficzna RYSIK", 40, 20);
        g.drawString("ul. Polna 40, Suwałki", 40, 40);
        g.drawString("tel. +48 505 100 000, faks. +48 87 923 12 12", 40, 60);
        // rysuj linię
        g.drawLine(40, 80, 360, 80);
    }
}

<html>
<head>
<title>Pracownia graficzna RYSIK</title>
</head>
<body bgcolor="green">
<center>
<h2>Polecamy usługi naszej firmy!!</h2>
<applet
    code = "DaneKontaktowe.class"
    width = "400"
    height = "100">
</center>
</body>
</html>
```

Tworząc kod apletu należy zwrócić uwagę, iż nie wszystkie operacje mogą zostać wykonane⁸⁹. W szczególności mocno ograniczony jest dostęp do zasobów komputera (pamięć masowa, urządzenia peryferyjne). Podyktowane jest to względami bezpieczeństwa systemu komputerowego użytkownika.

Pytania sprawdzające

1. Czym jest aplet? W jakim środowisku jest uruchamiany?
2. Jaka jest różnica pomiędzy apletem, a aplikacją komunikującą się poprzez konsolę?
3. Wymień zasadnicze elementy struktury dokumentu HTML.
4. Jaką funkcję pełni opcjonalny element PARAM występujący w strukturze dokumentu HTML.
5. Opisz sposób przekazywania do apletu wartości numerycznych.
6. Wskaż różnice występujące pomiędzy klasą `java.awt.Applet`, a `javax.swing.JApplet`? Kiedy stosowana jest każda z nich?
7. W jakich jednostkach wyrażane są wymiary apletu?

⁸⁹ Zobacz: http://java.sun.com/docs/books/tutorial/deployment/applet/security_practical.html

8. Zapoznaj się z dostępnymi informacjami dotyczącymi klasy `java.awt.Color`. Co oznacza akronim RGB?
9. Wymień metody umożliwiające rysowanie figur na płaszczyźnie. W jakim pakiecie się one znajdują?
10. Odszukaj w API Javy, w której klasie została zdefiniowana metoda `repaint()`. Jakie jest jej działanie?

Zadania do wykonania

POZYCJONOWANIE TEKSTU

Zadanie 156 – *PozycjonowanieTekstu.java*

Utwórz aplet zawierający nazwę szkoły/uczelni, do której uczęszczasz.

Rozwiązanie

Utwórz i skompiluj poniższy kod programu⁹⁰:

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

public class PozycjonowanieTekstu extends Applet {

    int szerokosc, wysokosc;

    public void init() {
        setBackground( Color.blue );
    }

    public void paint( Graphics g ) {
        Font font = new Font("SansSerif", Font.BOLD, 24);
        g.setFont(font);
        g.setColor( Color.yellow );
        g.drawString("Uniwersytet Ekonomiczny w Krakowie", 100, 100);
    }
}
```

Następnie utwórz plik `MojaUczelnia.html` zawierający strukturę dokumentu HTML, w której występuje odwołanie do programu `PozycjonowanieTekstu.class`. Otwórz plik HTML w przeglądarce internetowej.

```
<html>
<head>
  <title>Moja uczelnia</title>
</head>
<body>
  <center>
    <applet
      code = "PozycjonowanieTekstu.class"
      width = "640"
      height = "200">
    </center>
  </body>
</html>
```

Zadanie 157 – *HTML2Aplet.java*

Zmodyfikuj program `PozycjonowanieTekstu.java` w taki sposób, aby możliwa była dowolna zmiana nazwy szkoły/uczelni bez konieczności każdorazowej, powtórnej kompilacji programu. Wykorzystaj element `PARAM`.

Rozwiązanie

Wprowadź zmiany w pliku HTML określające nazwę oraz wartość parametru:

⁹⁰ Pakiet `java.awt` (ang. *Abstract Window Toolkit*) to niezależny od platformy systemowej zestaw klas do projektowania aplikacji w środowisku graficznym.

```

<html>
  <head>
    <title>Moja uczelnia</title>
  </head>
  <body>
    <center>
      <applet
        code = "HTML2Aplet.class"
        width = "640"
        height = "200">
        <param name = "uczelnia" value = "Uniwersytet Ekonomiczny w Krakowie">
      </applet>
    </center>
  </body>
</html>

```

Następnie zapoznaj się z metodą `getParameter()` klasy `java.applet.Applet`. Zmodyfikuj kod programu, odczytując, a następnie wyświetlając wartość przekazanego parametru:

```

...
String nazwaUczelni = getParameter("uczelnia");
g.drawString(nazwaUczelni, 100, 100);
...

```

Zadanie 158 – Wizytowka.java

Utwórz aplet zawierający dane znajdujące się na typowej wizytówce (nazwa firmy, nazwisko i imię, stanowisko, adres, telefon). Wielkość apletu powinna być zbliżona do wielkości wizytówki. Wykorzystaj dostępne kroje czcionek oraz różne ich wielkości. Wykorzystaj kolory. Wymienione dane personalne przekaż jako parametry elementu `APPLET`.

Rozwiązanie

Zapoznaj się z metodą `getFontMetrics()` klasy `java.awt.Graphics`, a także z metodami klasy `java.awt.FontMetrics`, aby określić położenie tekstu niezależnie od wielkości wizytówki. Umieść nazwę firmy dokładnie na środku wizytówki. Dla wyznaczenia rozmiaru wizytówki możesz posłużyć się metodą `getSize()` dostępnej w klasie `java.awt.Component`, z której dziedziczy `java.applet.Applet` (zobacz również na rozwiązanie następnego zadania).

FIGURY GEOMETRYCZNE

Zadanie 159 – RysowanieLinii.java

Utwórz aplet zawierający kombinację linii na płaszczyźnie.

Rozwiązanie

Zapoznaj się z dostępnymi metodami klasy `java.awt.Graphics`, w szczególności z metodą `drawLine()`. Określ wymiary apletu na 640 x 480 pikseli.

```

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

public class RysowanieLinii extends Applet {

    int szerokosc, wysokosc;

    public void init() {
        szerokosc = getSize().width;
        wysokosc = getSize().height;
        setBackground( Color.black );
    }

    public void paint( Graphics g ) {
        g.setColor( Color.yellow );
        for ( int i = 0; i <= 30; ++i ) {
            g.drawLine( szerokosc / 4, wysokosc, i * szerokosc / 30, 0 );
        }
        g.setColor( Color.green );
        for ( int i = 0; i <= 30; ++i ) {
            g.drawLine( 3 * szerokosc / 4, wysokosc, i * szerokosc / 30, 0 );
        }
    }
}

```

Zadanie 160 – ZbiorFigurGeometrycznych.java

Niech zmienne a , b oraz r posiadają wartości odpowiednio: 20, 40, 30. Utwórz aplet zawierający figury geometryczne (kwadrat, prostokąt oraz okrąg) o wymiarach określonych przez wartości podanych zmiennych. Dokonaj takiego rozmieszczenia figur, aby się wzajemnie nie pokrywały. Wartości a , b , r przekaz jako parametry apletu.

Rozwiązanie

Zapoznaj się z metodami dostępnymi w klasie `java.awt.Graphics`.

Zadanie 161 – Szachownica.java

Utwórz aplet reprezentujący szachownicę składającą się z 64 pól (8 x 8).

Zadanie 162 – KolaOlimpijskie.java

Utwórz aplet o wymiarach 200 x 100 pikseli, który przedstawia koła olimpijskie. Zastosuj właściwe kolory.

Zadanie 163 – FiguryGeometryczneLosowo.java

Utwórz aplet zawierający 5 prostokątów o rozmiarach i położeniu określonym w sposób losowy. Każda z figur powinna zostać wyróżniona odmiennym kolorem.

Zadanie 164 – BanerReklamowy.java

Zaprojektuj baner reklamowy o wymiarach 400 x 200 pikseli, a następnie utwórz aplet będący implementacją banera. Projekt powinien zawierać zarówno tekst, jak i elementy graficzne. Tekst banera przekaz jako wartość parametru.

WYKRESY

Zadanie 165 – WykresFunkcjiLiniowej.java

Poniższy kod programu zawiera wykres funkcji liniowej $y=x+50$. Rozbuduj program o wykresy następujących funkcji: $y=x/2+20$, $y=x/2-30$, $y=2x+20$, $y=x^2/200$. Każdą z funkcji wyróżnij innym kolorem.

```

<html>
<head></head>
<body>
<center>
<applet
    code = "WykresFunkcjiLiniowej.class"
    width = "500"
    height = "500">
</center>
</body>
</html>

```

```

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

public class WykresFunkcjiLiniowej extends Applet {

    int szerokosc, wysokosc;

    public void init() {
        szerokosc = getSize().width;
        wysokosc = getSize().height;
    }

    public void paint( Graphics g ) {

        // siatka współrzędnych
        g.setColor( Color.pink );
        for (int i=10; i < szerokosc; i+=10){
            g.drawLine(i,0,i,wysokosc);
        }
        for (int i=10; i < wysokosc; i+=10){
            g.drawLine(0, i, szerokosc, i);
        }
        // osie współrzędnych
        g.setColor( Color.black );
        g.drawLine(0, wysokosc/2, szerokosc, wysokosc/2);
        g.drawLine(szerokosc/2, 0, szerokosc/2, wysokosc);

        rysujFunkcjeLiniowa(g);
    }

    // funkcja liniowa postaci: y = x + 50
    public void rysujFunkcjeLiniowa(Graphics g){
        int x, y, a, b;
        g.setColor( Color.blue );
        for (x = -szerokosc/2; x < szerokosc/2 + 50; x++) {
            y = x + 50;
            a = x + szerokosc/2;
            b = szerokosc/2 - y;
            g.drawLine(a, b, a, b);
        }
    }
}

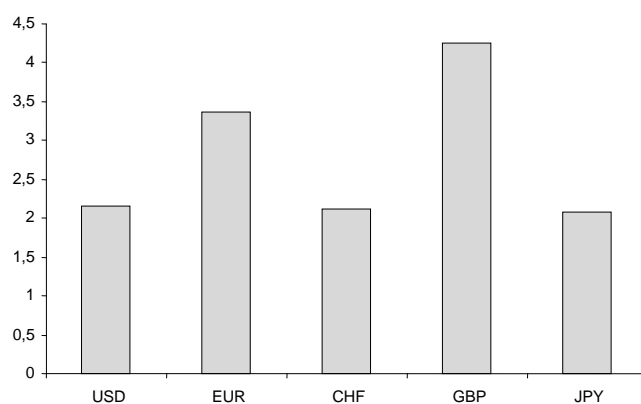
```

Zadanie 166 – PowierzchniaZiemi.java

Powierzchnia całkowita Ziemi wynosi 510 mln km². 71% tej wartości stanowi powierzchnia wodna, pozostałą część (29%) powierzchnia lądowa. Przedstaw na wykresie kołowym podane wartości.

Zadanie 167 – NotowaniaKursowWalut.java

Kursy podstawowych walut w stosunku do PLN w dniu 02-06-2008 kształtowały się następująco: 1 USD – 2,16 PLN, 1 EUR – 3,37 PLN, 1 CHF – 2,11 PLN, 1 GBP – 4,25 PLN, 100 JPY – 2,07 PLN. Poniższy wykres przedstawia graficzną reprezentację podanych notowań.



Rys. 11. Notowania kursów walut.

Utwórz aplet, który przedstawi graficznie notowania wybranych kursów walut, jak na załączonym wykresie. Wartości kursów prześlij jako parametry apletu.

Rozdział 10

Elementy graficznego interfejsu użytkownika

Cel jednostki

Po zrealizowaniu materiału będziesz w stanie:

- tworzyć aplikacje zawierające graficzny interfejs użytkownika,
- wykorzystywać standardowe kontenery oraz komponenty,
- implementować obsługę zdarzeń,
- umieszczać elementy graficzne w tworzonej aplikacji.

Wprowadzenie do zagadnień

Wykorzystanie w projektowanych aplikacjach graficznego interfejsu użytkownika GUI (ang. *Graphical User Interface*) zdecydowanie poprawia komfort użytkowania programu. Dostępny w Javie zbiór klas bibliotecznych Swing⁹¹ jest rozwinięciem istniejącej biblioteki AWT (ang. *Abstract Window Toolkit*), realizującej operacje graficzne w poprzednich wersjach języka. Zawarte w dystrybucji Javy pakiety graficzne składają się z pokaźnej liczby komponentów, które umieszczone w odpowiednich kontenerach, przy zastosowaniu obsługi zdarzeń, tworzą końcową aplikację. Poszczególne elementy wchodzące w skład aplikacji dostępne są w pakietach `javax.swing`, `java.awt` oraz `java.awt.event`.

Podstawą każdej aplikacji jest wybór odpowiedniego kontenera głównego (okna) służącego do przechowywania pozostałych elementów⁹². Swing oferuje kilka takich kontenerów: `JApplet`, `JDialog`, `JWindow` oraz `JFrame`, dostępnych w pakiecie `javax.swing`. Zazwyczaj wykorzystywana jest klasa `JFrame` udostępniająca definicję okna wraz z paskiem tytułowym i niezbędnymi przyciskami.

GLÓWNE OKNO APLIKACJI

Tworzenie interfejsu graficznego, przy zastosowaniu biblioteki Swing, sprowadza się do wykorzystania dostępnych w pakiecie zestawów klas. Utworzenie nowego okna polega na utworzeniu instancji klasy `javax.swing.JFrame`, ustaleniu jego rozmiaru oraz wyświetleniu go na ekranie⁹³.

⁹¹ Swing występuje również dość często pod nazwą JFC (ang. *Java Foundation Classes*).

⁹² Kontenery główne (okna) nazywane są inaczej komponentami ciężkimi (odwołują się do natywnych komponentów danej platformy systemowej).

⁹³ Domyślnie obiekty klasy `JFrame` nie są widoczne na ekranie.

```
import javax.swing.*;

public class ModelRGB {

    public static void main(String[] args) {

        // utworzenie okna wraz z określeniem tytułu
        JFrame okno = new JFrame("Model RGB przestrzeni barw");

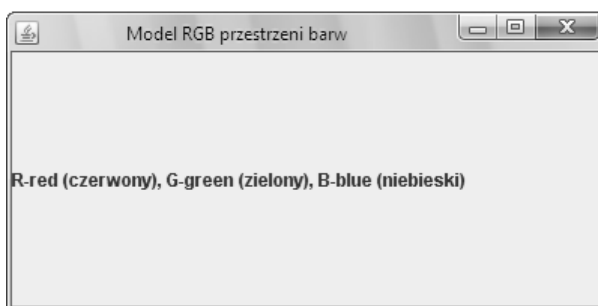
        // ustalenie rozmiarów
        okno.setSize(400, 200);

        // określenie operacji realizowanej podczas zamknięcia okna
        okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // wyświetlenie okna
        okno.setVisible(true);
    }
}
```

W tak utworzonym oknie można umieszczać poszczególne komponenty. Umieszczenie przykładowego tekstu (obiekt klasy `JLabel`) realizowane jest za pomocą instrukcji:

```
JLabel opisRGB =
    new JLabel("R-red (czerwony), G-green (zielony), B-blue (niebieski)");
okno.add(opisRGB);
```



Rys. 12. Okno aplikacji wykorzystującej graficzny interfejs użytkownika.

Wymienione instrukcje należy umieścić przed wyświetleniem okna. Taki sposób sprawdza się jedynie podczas umieszczania w oknie niewielkiej liczby komponentów.

KONTENERY

W przypadku większej liczby komponentów wykorzystywane są kontenery grupujące elementy. Jednym z nich jest obiekt klasy `JPanel`. Poniższy kod stanowi przykład aplikacji wyświetlającej okno zawierające 3 przyciski. Należy zwrócić uwagę na odmienny sposób tworzenia okna głównego (klasa dziedzicząca z klasy `JFrame`).

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

class Kontener extends JPanel {

    private JButton przyciskR, przyciskG, przyciskB;

    public Kontener() {

        // utwórz przyciski
        przyciskR = new JButton("R-czerwony");
        przyciskG = new JButton("G-zielony");
        przyciskB = new JButton("B-niebieski");

        // umieść przyciski w kontenerze
        add(przyciskR);
        add(przyciskG);
    }
}
```

```

        add(przyciskB);

        // ustal kolor tła kontenera
        setBackground(Color.green);
    }
}

class Okno extends JFrame {
    public Okno() {
        super("Model RGB przestrzeni barw");
        setSize(400, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

public class ModelRGBKontener {

    public static void main(String[] args) {

        // utwórz okno oraz kontener z przyciskami
        Okno okno = new Okno();
        Kontener kontener = new Kontener();

        // umieść kontener w oknie
        okno.add(kontener);

        // wyświetl okno
        okno.setVisible(true);
    }
}

```



Rys. 13. Grupowanie elementów w kontenerach.

Poza klasą `JPanel` istnieje szereg kontenerów umożliwiających grupowanie komponentów⁹⁴. Poniższa tabela przedstawia najczęściej wykorzystywane.

Tabela 4. Wykaz najczęściej stosowanych kontenerów.

KONTENER	REALIZOWANA FUNKCJA
<code>JPanel</code>	Grupowanie komponentów
<code>JScrollPane</code>	Dodawanie pasków przesuwnych
<code>JSplitPane</code>	Podział kontenera (w pionie bądź w poziomie)
<code>JTabbedPane</code>	Tworzenie zakładek

⁹⁴ Szczegółowa charakterystyka kontenerów oraz komponentów dostępna jest na stronie <http://java.sun.com/docs/books/tutorial/ui/features/components.html>.

JToolBar	Obsługa paska narzędziowego
----------	-----------------------------

KOMPONENTY

GUI udostępnia szereg komponentów realizujących funkcję interakcji z użytkownikiem. Dzięki ich wykorzystaniu użytkownik programu może wprowadzać informacje tekstowe, dokonywać wyboru poprzez wskazanie wyświetlonych opcji, czy też zatwierdzać wprowadzone dane. Biblioteka Swing dysponuje klasami realizującymi wymienione czynności. Wykaz najczęściej wykorzystywanych komponentów⁹⁵ zawarty został w poniższej tabeli.

Tabela 5. Wykaz wybranych komponentów pakietu Swing.

KOMPONENT	REALIZOWANA FUNKCJA
JButton	Przycisk
JComboBox	Lista rozwijana
JList	Lista
JCheckBox	Pole wyboru
JMenu	Menu
JRadioButton	Pole opcji
JSlider	Suwak
TextField, JTextArea	Pole tekstowe
JLabel	Etykieta tekstowa
JProgressBar	Wskaźnik postępu
JToolTip	Podpowiedź
JColorChooser	Panel wyboru koloru
JTable	Tabela danych
JTree	Układ hierarchiczny danych (struktura drzewa)

MENADŻERY ROZKŁADU

Rozmieszczenie poszczególnych elementów (kontenerów, czy też komponentów) w oknie aplikacji zależy od zastosowania tzw. menadżera rozkładu. Wpływa on zarówno na pozycję elementów względem siebie, jak również na ich rozmiar i sposób wyrównywania. Każdy kontener stosuje domyślny rozkład elementów. Jego zmiana możliwa jest poprzez użycie metody `setLayout()`. Poniższa tabela przedstawia typowe układy elementów zależne od zastosowanego menadżera rozkładu.

⁹⁵ Komponenty te nazywane są dość często komponentami lekkimi.

Tabela 6. Wybrane metody rozmieszczania elementów interfejsu.⁹⁶

MENADŻER	UKŁAD ELEMENTÓW
BorderLayout	wg kierunków geograficznych (północ, południe, wschód, zachód, środek)
BoxLayout	w rzędzie lub kolumnie
FlowLayout	sąsiadujący w poziomie
GridLayout, GridBagLayout	tabelaryczny
GroupLayout	w grupach
CardLayout	w kartach

Utworzenie okienka korzystającego z rozkładu BorderLayout oraz umieszczenie przykładowych komponentów może wyglądać w następujący sposób:

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

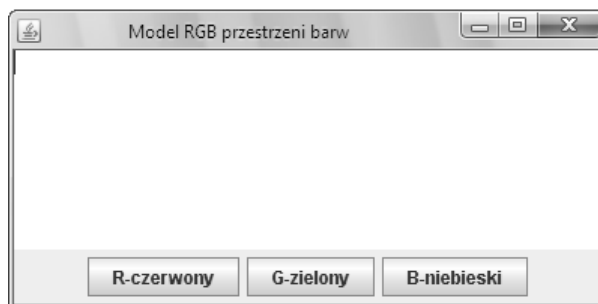
public class ModelRGBMenadzerRozkladu extends JFrame {

    public ModelRGBMenadzerRozkladu() {
        super("Model RGB przestrzeni barw");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel kontener = new JPanel();
        kontener.add(new JButton("R-czerwony"));
        kontener.add(new JButton("G-zielony"));
        kontener.add(new JButton("B-niebieski"));
        add(kontener, BorderLayout.SOUTH);
        add(new JTextArea(), BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        ModelRGBMenadzerRozkladu okno = new ModelRGBMenadzerRozkladu();
        okno.setVisible(true);
    }
}
```

⁹⁶ Szczegółowa charakterystyka menadżerów rozkładu umożliwiających rozmieszczenie poszczególnych elementów dostępna jest na stronie <http://java.sun.com/docs/books/tutorial/uiswing/layout/index.html>



Rys. 14. Rozmieszczanie elementów w oknie aplikacji.

OBSŁUGA ZDARZEŃ

Programowanie aplikacji wykorzystującej GUI związane jest nierozdzielnie z pojęciem obsługi zdarzeń. Generowane są one najczęściej przez mysz, klawiaturę, czy też poszczególne elementy interfejsu graficznego. W przypadku Javy zdarzeniem jest obiekt, który opisuje zmianę stanu swojego źródła spowodowaną np. przez naciśnięcie klawisza, kliknięcie myszką, wybór elementu z listy wyboru, zamknięcie okna. Źródłem jest obiekt, który wygenerował zdarzenie.

Obsługa zdarzeń sprowadza się do deklaracji klasy obsługującej konkretny typ zdarzenia oraz powiązania obiektu tej klasy z konkretnym komponentem (źródłem zdarzenia). Wszystkie zdarzenia określone są za pomocą specjalnych interfejsów (tzw. nasłuchiwalcy).

Tabela 7. Obsługa zdarzeń⁹⁷.

INTERFEJS	RODZAJ ZDARZENIA
<code>ActionListener</code>	zmiana stanu komponentu
<code>WindowListener</code>	zmiana stanu okna
<code>MouseListener</code>	zmiana stanu przycisków myszy
<code>MouseMotionListener</code>	zmiana położenia wskaźnika myszy
<code>ComponentListener</code>	zmiana widoczności, rozmiarów i położenia komponentu
<code>FocusListener</code>	aktywacja komponentu
<code>ListSelectionListener</code>	wybór wartości z listy

Każdy interfejs zawiera nagłówki metod reprezentujące poszczególne rodzaje zdarzeń. Klasa obsługująca zdarzenia implementuje odpowiedni interfejs (interfejsy). Poniżej podany został przykładowy kod klasy obsługującej kliknięcie przycisku.

⁹⁷ Pełny opis obsługi zdarzeń wraz z przykładowymi kodami źródłowymi dostępny jest na stronie <http://java.sun.com/docs/books/tutorial/uiswing/events/index.html>

```

class Obsluga implements ActionListener {

    public void actionPerformed(ActionEvent e){

        // instrukcje wykonywane podczas wystąpienia zdarzenia
        // ...

    }
}

```

Powiązanie komponentu z obiektem obsługującym zdarzenie realizowane jest za pomocą metody `addNazwaInterfejsu()`⁹⁸.

```

JButton przycisk = new JButton("Tekst przycisku");
Obsluga dzialanie = new Obsluga();

przycisk.addActionListener(dzialanie);

```

Wszystkie interfejsy i klasy związane z obsługą zdarzeń dostępne są w pakiecie `java.awt.event`. Poniższy kod programu zawiera przykładową aplikację, w której zaimplementowana została obsługa zdarzenia polegającego na naciśnięciu przycisku umieszczonego w oknie aplikacji. Rezultatem tej czynności jest zmiana koloru wyświetlanego okna.

```

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

public class ModelRGBObslugaZdarzen extends JFrame {
    private JButton przyciskR, przyciskG, przyciskB;
    private JPanel panel1, panel2;

    public ModelRGBObslugaZdarzen() {
        super("Model RGB przestrzeni barw");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        panel1 = new JPanel();
        panel2 = new JPanel();

        przyciskR = new JButton("R-czerwony");
        przyciskG = new JButton("G-zielony");
        przyciskB = new JButton("B-niebieski");

        ObslugaZdarzenia zmianaNaCzerwony = new ObslugaZdarzenia(Color.red);
        ObslugaZdarzenia zmianaNaZielony = new ObslugaZdarzenia(Color.green);
        ObslugaZdarzenia zmianaNaNiebieski = new ObslugaZdarzenia(Color.blue);

        przyciskR.addActionListener(zmianaNaCzerwony);
        przyciskG.addActionListener(zmianaNaZielony);
        przyciskB.addActionListener(zmianaNaNiebieski);

        panel1.add(przyciskR);
        panel1.add(przyciskG);
        panel1.add(przyciskB);

        panel2.setBackground(Color.white);

        add(panel1, BorderLayout.SOUTH);
        add(panel2, BorderLayout.CENTER);
    }

    private class ObslugaZdarzenia implements ActionListener {
        private Color kolor;

        public ObslugaZdarzenia(Color k){
            this.kolor = k;
        }

        public void actionPerformed(ActionEvent e) {
            panel2.setBackground(kolor);
        }
    }
}

```

⁹⁸ NazwaInterfejsu oznacza konkretny typ zdarzenia.

```

        repaint();
    }
}

public static void main(String[] args) {
    ModelRGBObslugaZdarzen okno = new ModelRGBObslugaZdarzen();
    okno.setVisible(true);
}
}

```

ELEMENTY GRAFICZNE

Każdy obiekt graficznego interfejsu użytkownika (kontener, komponent) posiada metodę `paintComponent()`, zadaniem której jest narysowanie obiektu na ekranie monitora. Przesłonięcie tej metody umożliwia umieszczenie przez programistę dodatkowych elementów graficznych (punkty, linie, figury geometryczne) w wybranym obiekcie. Mimo, iż operacje rysowania przeprowadzić można na każdym komponencie dziedziczącym z klasy `JComponent` to najczęściej wykorzystywany jest kontener `JPanel`. Tworzone aplikacje graficzne wykorzystują przeważnie dostępne klasy zawarte w pakiecie `java.awt.geom`⁹⁹, jak również klasy `java.awt.Color`, czy `java.awt.Font`.

Poniższa aplikacja rysuje 3 prostokąty, wypełniając każdy z nich jednym z kolorów palety RGB. Każdy z kolorów został opisany poprzez umieszczenie w nim ciągu znaków.

```

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;

class MojPanel extends JPanel {

    private int x, y;
    private int wysokoscCzcionki, szerokoscNapisu;
    private String czerwony = "R - czerwony";
    private String zielony = "G - zielony";
    private String niebieski = "B - niebieski";

    public MojPanel () {
        setBackground(Color.white); // ustalenie koloru tła
    }

    public void paintComponent(Graphics g) {

        x = getSize().width;
        y = getSize().height;

        super.paintComponent(g); // wywołanie metody paintComponent() z klasy nadrzędnej
        Graphics2D g2 = (Graphics2D) g; // rzutowanie parametru

        g2.setPaint(Color.red); // ustawienie koloru rysowania
        g2.fillRect(0, 0, x, y/3);

        g2.setPaint(Color.green); // ustawienie koloru rysowania
        g2.fillRect(0, y/3, x, y/3);

        g2.setPaint(Color.blue); // ustawienie koloru rysowania
        g2.fillRect(0, y/3*2, x, y/3);

        Font czcionka = new Font("Verdana", Font.PLAIN, 20);
        g2.setFont(czcionka); // ustawienie czcionki

        g2.setPaint(Color.white);

        wysokoscCzcionki = getFontMetrics(czcionka).getHeight();

        // wyświetlenie napisu
        szerokoscNapisu = getFontMetrics(czcionka).stringWidth(czerwony);
        g2.drawString(czerwony, x/2-szerokoscNapisu/2, (y/3)/2+(wysokoscCzcionki/2));

        // wyświetlenie napisu
        szerokoscNapisu = getFontMetrics(czcionka).stringWidth(zielony);
        g2.drawString(zielony, x/2-szerokoscNapisu/2, (y/3)+(wysokoscCzcionki/2));
    }
}

```

⁹⁹ Przykładowe klasy umożliwiające rysowanie punktów, linii, czy figur geometrycznych na płaszczyźnie to: `Point2D`, `Line2D`, `Rectangle2D`, `Ellipse2D`.

```

        // wyświetlenie napisu
        szerokoscNapisu = getFontMetrics(czcionka).stringWidth(niebieski);
        g2.drawString(niebieski, x/2-szerokoscNapisu/2,
            (y-((y/3)/2)+(wysokoscCzcionki/2)));
    }
}

public class ModelRGBFigury {

    public static void main(String[] args) {

        JFrame ramka = new JFrame("Model RGB");
        MojPanel panel = new MojPanel();

        ramka.add(panel);

        ramka.setSize(400, 200);
        ramka.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ramka.setVisible(true);
    }
}

```



Rys. 15. Wykorzystanie elementów grafiki użytkownika.

Pytania sprawdzające

1. Wskaż cel stosowania pakietu Swing?
2. Do czego wykorzystywane są podstawowe kontenery (okna)?
3. Scharakteryzuj stosowaną konwencję nazewnictwa klas w pakiecie Swing.
4. Wymień przykładowe nazwy kontenerów umożliwiające grupowanie obiektów.
5. Podaj różnice występujące pomiędzy kontenerami JPanel oraz JScrollPane.
6. Wymień komponenty pakietu Swing stosowane przy tworzeniu graficznego interfejsu użytkownika.
7. Jaki komponent odpowiada za etykiety tekstowe? Czy mogą one posiadać ikony?
8. Czym są oraz jaką funkcję pełnią menadżery rozkładu? Wymień ich nazwy.
9. Jaka metoda służy do zmiany domyślnego sposobu rozmieszczenia elementów?
10. Czy pakiet Swing umożliwia obsługę zdarzeń?
11. Opisz zadanie interfejsów ActionListener oraz MouseListener.
12. Jak powinna zostać zbudowana klasa obsługująca występujące w programie zdarzenia?
13. W jaki sposób odbywa się powiązanie komponentu ze zdarzeniem i jego obsługą?
14. Która metoda powinna zostać przesłonięta, aby możliwe było wykonanie operacji rysowania?
15. Wymień klasy stosowane w aplikacjach korzystających z grafiki.

Zadania do wykonania

Zadanie 168 – ModelRGBPasekStanu.java

Uzupełnij kod aplikacji ModelRGB, zawarty w części teoretycznej rozdziału, w taki sposób, aby nazwa prezentowanego koloru wyświetlana była na pasku statusu znajdującego się w dolnej części okna.

Rozwiązanie

Uruchom poniższy kod programu. Zwróć uwagę na sposób odczytu informacji o aktualnym kolorze, gdzie wykorzystane zostało dodatkowe pole w wewnętrznej klasie Obsługa. Odszukaj w programie te fragmenty kodu, które odpowiedzialne są za wyświetlanie nazwy koloru na pasku statusu (komponent JLabel).

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

public class ModelRGBPasekStanu extends JFrame {
    private JButton przyciskR, przyciskG, przyciskB;
    private JPanel panel1, panel2;
    private JLabel aktualnyStatus;

    public ModelRGBPasekStanu() {
        super("Model RGB przestrzeni barw");
        setSize(350, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        panel1 = new JPanel();
        panel2 = new JPanel();
        aktualnyStatus = new JLabel("Aktualny kolor: biały");

        przyciskR = new JButton("R-czerwony");
        przyciskG = new JButton("G-zielony");
        przyciskB = new JButton("B-niebieski");

        Obsluga zmienNaCzerwony = new Obsluga(Color.red, "czerwony");
        Obsluga zmienNaZielony = new Obsluga(Color.green, "zielony");
        Obsluga zmienNaNiebieski = new Obsluga(Color.blue, "niebieski");

        przyciskR.addActionListener(zmienNaCzerwony);
        przyciskG.addActionListener(zmienNaZielony);
        przyciskB.addActionListener(zmienNaNiebieski);

        panel1.add(przyciskR);
        panel1.add(przyciskG);
        panel1.add(przyciskB);

        panel2.setBackground(Color.white);

        add(panel1, BorderLayout.NORTH);
        add(panel2, BorderLayout.CENTER);
        add(aktualnyStatus, BorderLayout.SOUTH);
    }

    private class Obsluga implements ActionListener {
        private Color kolor;
        private String napis;

        public Obsluga(Color k, String n) {
            this.kolor = k;
            this.napis = n;
        }

        public void actionPerformed(ActionEvent e) {
            panel2.setBackground(kolor);
            aktualnyStatus.setText("Aktualny kolor: " + napis);
            repaint();
        }
    }

    public static void main(String[] args) {
        ModelRGBPasekStanu okno = new ModelRGBPasekStanu();
    }
}
```

```

        okno.setVisible(true);
    }
}

```

Zadanie 169 – ModelCMYK.java

Odszukaj w sieci Internet, jakie barwy wchodzi w skład modelu CMYK przestrzeni barw. Wzorując się na aplikacji ModelRGB napisz program, który realizował będzie zbliżoną funkcjonalność dla 4 kolorów wchodzących w skład modelu CMYK.

Zadanie 170

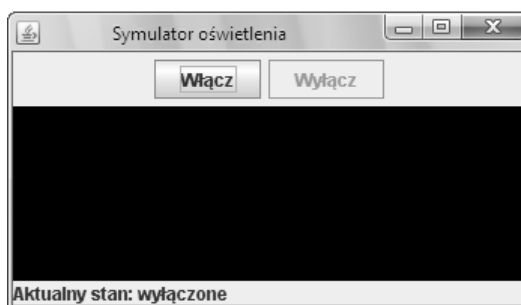
Udostępnione przez firmę SUN materiały w wersji elektronicznej zawarte na stronach internetowych pod adresem <http://java.sun.com/docs/books/tutorial/ui/features/components.html> zawierają szczegółowy opis wykorzystania dostępnych kontenerów i komponentów biblioteki graficznej Swing. Zapoznaj się ze sposobem użycia komponentów: JComboBox, JRadioButton, JTextField, JMenu, JDialog. Skopiuj znajdujące się tam przykładowe aplikacje. Skompiluj je, uruchom i porównaj ich działanie ze znajdującym się na stronach internetowych opisem działania komponentu.

Zadanie 171 – SymulatorOswietlenia.java

Zadaniem aplikacji jest symulacja oświetlenia w pomieszczeniu. Włączenie lub wyłączenie oświetlenia realizowane jest przy pomocy dwóch przycisków co powoduje zmianę koloru w oknie (czarny – oświetlenie wyłączone, żółty – oświetlenie włączone). Przykładowa postać aplikacji została przedstawiona na poniższym rysunku. Zwróć uwagę, iż nie jest możliwe włączenie oświetlenia, gdy jest już ono włączone, ani jego wyłączenie, gdy w pomieszczeniu jest ciemno (nieaktywne przyciski).

Rozwiązanie

Zapoznaj się z dokumentacją klasy JButton.



Rys. 16. Symulator oświetlenia.

Zadanie 172 – KonwerterWalutGUI.java

Odszukaj w sieci Internet kurs wymiany dla następujących walut: USD, EUR, PLN. Utwórz aplikację, zadaniem której będzie przeliczenie dowolnej kwoty wyrażonej w wybranej walucie na inną, przy wykorzystaniu kursu wymiany. Uwzględnij możliwość modyfikacji wprowadzonych kursów wymiany w sytuacji jego zmiany na rynku.

Zadanie 173 – SumatorLiczbyRzeczywistych.java

Utwórz aplikację, zadaniem której jest wyznaczenie sumy wartości dwóch liczb typu rzeczywistego.

Rozwiązanie

Skompiluj i uruchom poniższy kod programu. Następnie dokonaj jego modyfikacji, aby wykonanie operacji możliwe było jedynie w przypadku wprowadzenia poprawnych wartości typu rzeczywistego.

```

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

public class SumatorLiczbyRzeczywistych extends JFrame {
    private JButton przycisk;
    private JTextField liczbaPierwsza, liczbaDruga;
    private JLabel wynik;
}

```



```

public SumatorLiczBzRzeczywistych() {
    super("Sumator liczb rzeczywistych");
    setSize(350, 70);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panel = new JPanel();
    liczbaPierwsza = new JTextField("0", 4);
    liczbaDruga = new JTextField("0", 4);
    JLabel plus = new JLabel("+");
    wynik = new JLabel("");

    przycisk = new JButton("=");

    Obsluga dzialanie = new Obsluga();

    przycisk.addActionListener(dzialanie);

    panel.add(liczbaPierwsza);
    panel.add(plus);
    panel.add(liczbaDruga);
    panel.add(przycisk);
    panel.add(wynik);

    add(panel);
}

private class Obsluga implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        Float wartoscPierwsza = new Float(liczbaPierwsza.getText());
        Float wartoscDruga = new Float(liczbaDruga.getText());
        float suma = wartoscPierwsza + wartoscDruga;
        wynik.setText("" + suma);
    }
}

public static void main(String[] args) {
    SumatorLiczBzRzeczywistych okno = new SumatorLiczBzRzeczywistych();
    okno.setVisible(true);
}
}

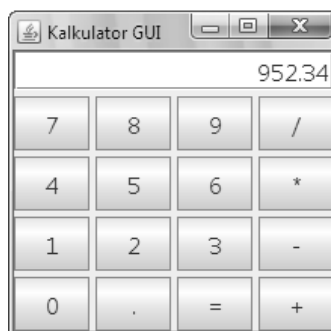
```

Zadanie 174 – Kalkulator.java

Zmodyfikuj kod aplikacji `SumatorLiczBzRzeczywistych.java`, aby możliwe było wykonywanie czterech podstawowych operacji arytmetycznych (dodawanie, odejmowanie, mnożenie, dzielenie).

Zadanie 175 – KalkulatorGUI.java

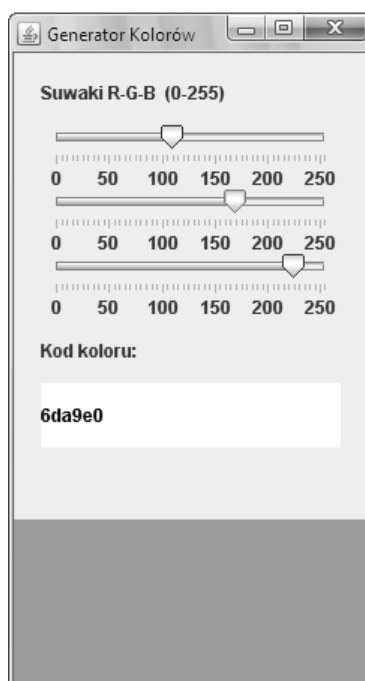
Wykorzystując menadżery rozkładu `BorderLayout` i `GridLayout` oraz stosowne komponenty (pole tekstowe, przyciski) utwórz model kalkulatora przedstawiony na poniższym rysunku. Dodaj obsługę zdarzeń, aby możliwe było wykonywanie podstawowych operacji arytmetycznych.



Rys. 17. Kalkulator GUI.

Zadanie 176 – GeneratorKolorow.java

Znajomość wartości kolorów wyrażonych przy użyciu palety barw RGB niezbędna jest w szczególności programistom zajmującym się projektowaniem stron internetowych. Utwórz aplikację, która prezentowała będzie dowolny kolor wyrażony za pomocą trzech liczb całkowitych z przedziału 0..255. Wartości te powinny zostać ustalone poprzez zmianę położenia 3 suwaków, jak pokazano na rysunku poniżej. Wyświetl w oknie aplikacji kod koloru w wymaganym przez język HTML zapisie szesnastkowym (np. kolor czerwony: #FF0000).



Rys. 18. Generator kolorów.

Zadanie 177 – Lokata.java

Procent składany oznacza sposób oprocentowania wkładu pieniężnego K , polegającego na tym, że roczny dochód w postaci odsetek jest doliczany do kapitału i procentuje z nim w latach następnych. Można wyrazić to wzorem: $K_n = K \cdot (1 + r/100\%)^n$, gdzie r oznacza stopę procentową, a n liczbę lat trwania lokaty. Napisz program realizujący opisaną funkcjonalność. Następnie za jego pomocą wyznacz wartości przyszłe lokat dla:

- $K = 1500\text{zł}$, $n = 10$ lat, $r = 7,5\%$
- $K = 7230\text{zł}$, $n = 15$ lat, $r = 6,3\%$
- $K = 2590\text{zł}$, $n = 30$ lat, $r = 8,0\%$

Zadanie 178 – PrzeglądarkaPlikowTekstowych.java

Utwórz aplikację umożliwiającą wyświetlenie zawartości wskazanego pliku tekstowego.

Rozwiązanie

Poniższy kod programu zawiera rozwiązanie zadania. Dokonaj jego kompilacji. Uruchom aplikację, a następnie odszukaj w kodzie programu te fragmenty, które odpowiedzialne są za realizację poszczególnych funkcji programu.

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

public class PrzeglądarkaPlikowTekstowych extends JFrame implements ActionListener {

    JMenuItem menuOtworz;
    JMenuItem menuZamknij;
    JTextArea poleTekstowe;
    File plik;

    /** Definicja wyglądu aplikacji */
    public PrzeglądarkaPlikowTekstowych() {
        super("Przeglądarka plików tekstowych");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // dodaj obszar wyświetlania pliku tekstowego
        setLayout(new BorderLayout());
        poleTekstowe = new JTextArea("");
        poleTekstowe.setEditable(false);
        JScrollPane panel = new JScrollPane(poleTekstowe);
        add(panel, BorderLayout.CENTER);

        // dodaj menu programu wraz obsługą zdarzeń
        JMenu menu = new JMenu("Plik");
        menuOtworz = new JMenuItem("Otwórz");
        menuOtworz.addActionListener(this);
        menuZamknij = new JMenuItem("Zamknij");
        menuZamknij.addActionListener(this);
        menu.add(menuOtworz);
        menu.add(menuZamknij);
        JMenuBar menuBar = new JMenuBar();
        menuBar.add(menu);
        setJMenuBar(menuBar);
        setSize(400, 400);
    }

    /** Obsługa zdarzeń (wybór opcji menu) */
    public void actionPerformed(ActionEvent e) {
        boolean status = false;

        String akcja = e.getActionCommand();

        // wybrano z menu opcję "Otwórz"
        if (akcja.equals("Otwórz")) {
            status = otworzPlik();
            if (!status)
                JOptionPane.showMessageDialog(null, "Błąd otwarcia pliku!", "Błąd",
                    JOptionPane.ERROR_MESSAGE);
        }
        // wybrano z menu opcję "Zamknij"
        else if (akcja.equals("Zamknij")) {
            System.exit(0);
        }
    }

    /** Odczytanie pliku tekstowego i umieszczenie jego zawartości w oknie programu */
    private boolean otworzPlik() {

        // wyświetl okno dialogowe z możliwością wyboru pliku tekstowego
        JFileChooser dialog = new JFileChooser();
        dialog.setDialogTitle("Otwórz plik");
        dialog.setFileSelectionMode(JFileChooser.FILES_ONLY);
        dialog.setCurrentDirectory(new File("."));

        // otwórz wskazany plik i odczytaj jego zawartość
```

```

int wynik = dialog.showOpenDialog(this);
if (wynik == JFileChooser.CANCEL_OPTION) {
    return true;
} else if (wynik == JFileChooser.APPROVE_OPTION) {

    plik = dialog.getSelectedFile();
    String zawartosc = odczytajPlik(plik);

    if (zawartosc != null)
        poleTekstowe.setText(zawartosc);
    else
        return false;
} else {
    return false;
}
return true;
}

/** Odczytanie zawartości pliku tekstowego */
private String odczytajPlik(File file) {
    StringBuffer bufor;
    String linia;

    try {
        FileReader in = new FileReader(file);
        BufferedReader bin = new BufferedReader(in);
        bufor = new StringBuffer();

        while ((linia = bin.readLine()) != null) {
            bufor.append(linia + "\n");
        }

        in.close();
    }
    catch(IOException e) {
        return null;
    }
    return bufor.toString();
}

/** uruchomienie programu */
public static void main(String[] args) {
    PrzegladarkaPlikowTekstowych przegladarka = new PrzegladarkaPlikowTekstowych();
    przegladarka.setVisible(true);
}
}

```

Zadanie 179 – ZaawansowanaPrzegladarkaPlikowTekstowych.java

Rozszerz działanie aplikacji PrzegladarkaPlikowTekstowych.java. Dodaj w menu programu opcje, realizujące funkcje:

- zamiany liter wielkich na małe, a małych na wielkie (opcja menu: „Zmień wielkość liter”),
- zmiany kierunku wyświetlania tekstu („Pokaż tekst wspak”),
- zwiększenia odległości poprzedzającej każdą linię 2 znaków spacji („Zwiększ wcięcie”),
- zmniejszenia odległości poprzedzającej każdą linię 2 znaków spacji („Zmniejsz wcięcie”).

Zadanie 180 – EdytorTekstu.java

Wykorzystaj kod aplikacji PrzegladarkaPlikowTekstowych.java do utworzenia znakowego edytora tekstu. Program powinien umożliwiać edycję dowolnie wybranego pliku tekstowego. Dodaj do programu funkcję tworzenia nowego pliku (opcja menu: „Nowy”) oraz funkcję zapisu (opcja menu: „Zapisz”).

Zadanie 181 – EdytorTekstuZPaskiemNarzedziowym.java

Uzupełnij edytor tekstu z zadania EdytorTekstu.java o pasek narzędziowy (toolbar). Dodaj przyciski, które realizować będą wszystkie funkcje dostępne w menu programu.

Zadanie 182 – RekrutacjaPracownikow.java

Dział spraw pracowniczych firmy „Fanga Motor” zajmuje się przetwarzaniem danych rekrutacyjnych kandydatów. Zaprojektuj aplikację, która umożliwi wprowadzanie poniższych informacji:

- imię (pole tekstowe),
- nazwisko (pole tekstowe),

płeć (kobieta, mężczyzna – pola opcji),
 rok urodzenia (pole tekstowe),
 wykształcenie (podstawowe, zawodowe, średnie, wyższe – lista rozwijana),
 krótkie umotywowanie swojej kandydatury (obszar tekstu),
 znajomość języków (angielski, niemiecki, francuski, hiszpański rosyjski – pola wyboru),
 telefon kontaktowy (pole tekstowe),
 adres e-mail (pole tekstowe).

Zaprojektuj formularz wykorzystując komponenty wymienione w nawiasach. Skorzystaj z menadżerów rozkładu. Zapisuj wprowadzane informacje o kandydatach do pliku tekstowego `RekrutacjaPracownikow.csv` (format CSV). Każdy wiersz powinien zawierać dane jednego kandydata. Przykładową zawartość pliku podano poniżej:

```
Jan,Nowak,M,1974,zawodowe,N,501100100,nowakj@gmail.com
Monika,Maj,K,1982,średnie,T,602555111,majm@onet.pl
```

Zadanie 183 – *KalkulatorKredytowy.java*

Kalkulator kredytowy pozwala na wyznaczenie kosztu kredytu oraz wielkości odsetek, przy założonym poziomie oprocentowania nominalnego, okresie, na jaki kredyt został zaciągnięty, częstotliwości kapitalizacji odsetek i sposobie spłat rat kredytu (raty równe lub malejące). Napisz program realizujący opisaną funkcjonalność. Utwórz interfejs użytkownika jak przedstawiono na rysunku poniżej. Algorytm obliczenia kosztu kredytu i wielkości odsetek znajdziesz w sieci Internet.

Rys. 19. Kalkulator kredytowy.

Zadanie 184 – *WykresKolowy.java*

Odszukaj w sieci Internet informacje dotyczące wielkości wpływów do budżetu państwa z tytułu podatków PIT, VAT oraz CIT za ubiegły rok podatkowy. Utwórz aplikację przedstawiającą graficznie wymienione wielkości. Zastosuj wykres kołowy. Dobierz kolory oraz umieść na wykresie niezbędne etykiety (tytuł wykresu, opisy pozycji).

Zadanie 185 – *KalkulatorKredytowyAplet.java*

Bazując na aplikacji `KalkulatorKredytowy.java`, utwórz aplet, którego zadaniem będzie wyświetlenie Kalkulatora Kredytowego w oknie przeglądarki internetowej.

Bibliografia

- Arnold K., Gosling J., Holmes D. [2005], *Java Programming Language*, Prentice Hall, New Jersey
- Arnold K., Gosling J., Holmes D. [2005], *Java(TM) Language Specification*, The (4th Edition), Prentice Hall, New Jersey
- Barteczko K., Drabik W., Starosta B. [2003], *Ćwiczenia z programowania w języku Java*, Mikom, Warszawa
- Darwin I. [2004], *Java Cookbook (Second Edition)*, O'Reilly Media, Inc.
- Eck D. [2006], *Introduction to Programming Using Java*, Fifth Edition, New York,
URL:<http://math.hws.edu/javanotes/index.html>, [dostęp 2008-02-14]
- Eckel B. [2002], *Thinking In Java*, 3rd Edition, URL:<http://www.mindviewinc.com/Books/> [dostęp 2008-02-14]
- Eckel B. [2006], *Thinking In Java*, Wydanie IV, Helion, Gliwice
- Horstmann C., Cornell G. [2003], *Core Java 2. Podstawy*, Helion, Gliwice
- Horstmann C., Cornell G. [2005], *Core Java 2. Techniki zaawansowane*, Helion, Gliwice
- Lis M. [2006], *Java. Ćwiczenia praktyczne*, Helion, Gliwice
- MacVittie D.W., MacVittie L.A. [1996], *Programowanie zorientowane obiektowo. Nowy sposób myślenia*, Mikom, Warszawa
- Meshplex, The Tutorial Database, *Introduction to Java*,
URL:http://www.meshplex.org/wiki/Java/Introduction_to_Java [dostęp 2008-02-14]
- Roy P.V., Haridi S. [2005], *Programowanie. Koncepcje, techniki i modele*, Helion, Gliwice
- Sedgewick R., Wayne K. [2007], *Introduction to Programming in Java: An Interdisciplinary Approach*, Addison Wesley
- Sun Microsystems, *Java Platform Standard Edition 6. API Specification*,
URL:<http://java.sun.com/javase/6/docs/api/> [dostęp 2008-02-14]
- Sun Microsystems, *Java tutorial*, URL:<ftp://ftp.javasoft.com/docs/tutorial.zip>, [dostęp 2009-01-24]
- Sun Microsystems, *The Java Tutorials*, URL: <http://java.sun.com/docs/books/tutorial/>, [dostęp 2008-02-14]
- Swartz F., *Java Basics*, URL:<http://www.leepoint.net/JavaBasics/>. [dostęp 2009-01-05]
- Swartz F., *Java Programming Notes*, URL:<http://www.leepoint.net/notes-java/>. [dostęp 2009-01-05]
- Wirth N. [2002], *Algorytmy + struktury danych = programy*, WNT, Warszawa
- Wrycza S., Marcinkowski B., Wyrzykowski K. [2006], *Język UML 2.0 w modelowaniu systemów informatycznych*, Helion, Gliwice
- Wu T. [2007], *A Comprehensive Introduction to Object-Oriented Programming with Java*, McGraw-Hill
- Zukowski J. [2005], *The Definitive Guide to Java Swing*, Third Edition, Apress

Spis rysunków

Rys. 1. Etapy tworzenia i uruchamiania aplikacji.	8
Rys. 2. Diagram UML klasy <code>Telefon</code>	51
Rys. 3. Tworzenie obiektów na podstawie klasy.	53
Rys. 4. Relacja kompozycji.....	63
Rys. 5. Relacja dziedziczenia (klasa <code>Telefon</code> oraz klasy pochodne).....	65
Rys. 6. Graficzna reprezentacja zależności między klasami.	75
Rys. 7. Interfejs i klasa implementująca.	78
Rys. 8. Struktura wyjątków.	90
Rys. 9. Wejściowe i wyjściowe strumienie danych.	93
Rys. 10. Określanie gabarytów apletu.....	103
Rys. 11. Notowania kursów walut.	109
Rys. 12. Okno aplikacji wykorzystującej graficzny interfejs użytkownika.	111
Rys. 13. Grupowanie elementów w kontenerach.	112
Rys. 14. Rozmieszczanie elementów w oknie aplikacji.....	115
Rys. 15. Wykorzystanie elementów grafiki użytkownika.	118
Rys. 16. Symulator oświetlenia.....	120
Rys. 17. Kalkulator GUI.	122
Rys. 18. Generator kolorów.	122
Rys. 19. Kalkulator kredytowy.	125

Spis tabel

Tabela 1. Modyfikatory dostępu do składowych obiektu.	41
Tabela 2. Przykładowy ranking osób.	61
Tabela 3. Klasy bazowe dla operacji wejścia-wyjścia.	93
Tabela 4. Wykaz najczęściej stosowanych kontenerów.	112
Tabela 5. Wykaz wybranych komponentów pakietu Swing.	113
Tabela 6. Wybrane metody rozmieszczania elementów interfejsu.	114
Tabela 7. Obsługa zdarzeń.	115