**Laboratorium 1**

Wstęp. Różnice pomiędzy C++ a Javą.

[Eclipse IDE | The Eclipse Foundation](https://eclipseide.org/)

**Wstęp do zajęć na UG**

**Trening przygotowawczy**

**Zadanie 0 - Przygotowanie środowiska**

Zainstalować/uruchomić środowisko eclipse na komputerach laboratoryjnych.

**Instalacja eclipse na kontach domowych w laboratorium komputerowym**

1. Otwieramy terminal: Application → System Tools → MATE Terminal
2. Instalacja właściwa:
   1. Wpisujemy w terminalu **eclipse-install**
   2. Przechodzimy przez proces instalacji podążając za kreatorem (wszystkie opcje domyślne!)
   3. **UWAGA!**Instalacja w laboratorium trwa długo. To normalne.
3. Środowisko eclipse uruchomiamy wpisując w terminalu **eclipse**
   1. **Przy pierwszym uruchomieniu środowiska i pierwszym tworzeniu projektu komputer będzie działał bardzo wolno.**To się poprawi samoistnie przy kolejnych uruchomieniach.

##### **Pierwszy program**

Pamiętacie najprostszy kod wypisujący napis na ekran w C++? Tak wygląda to w Javie:

|  |  |
| --- | --- |
| **Hello World: C++** | **Hello World:  Java** |
| #include <iostream>    **using** **namespace** std;    **int** main(**int** argc, **char**\*\* argv)  {  cout<<"Hello World!";  **return** 0;  } | **class** Hello  {  **public** **static** void main(**String**[ ] args)  {  **System**.out.println("Hello World!");  }  } |

|  |  |
| --- | --- |
| **Kompilacja: C++** | **Kompilacja: Java** |
| g++ [lista plików do skompilowania] | javac [lista plików do skompilowania] |

##### **Zadanie 1 - Java w notatniku i terminalu**

Utworzyć plik tekstowy o nazwie **Hello.java**i zawartości:

**class** Hello

{

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

{

**System**.out.println("Hello World!");

}

}

Następnie skompilować go za pomocą polecenia javac.

javac [lista plików do skompilowania]

czyli a naszym przypadku: **javac Hello.java**

I uruchomić:

java [nazwa klasy uruchamianej]

czyli a naszym przypadku:**java Hello**

##### **Eclipse**

Na zajęciach nie będziemy jednak pisać w notatniku, tylko korzystać ze środowiska **Eclipse**. Eclipse to bardzo popularne **zintegrowane środowisko programistyczne** (ang. **IDE**), czyli zespół aplikacji służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Pozwala na instalację dużej ilości różnych wtyczek, dopasowanych do potrzeb programisty.

Podstawowe informacje które mogą się przydać dzisiaj na zajęciach:

1. Po uruchomieniu zostaniesz zapytany o **workspace** - katalog, w którym będą zapisywane nowe projekty

2. Tworzenie**nowego projektu**w Eclipse

Wystarczy wybrać z menu: (File/New/Project... -> Java Project -> wybrać nazwę projektu -> Finish)

3. Dodanie do projektu **nowej klasy**

 File/New/Class -> wybrać nazwę klasy -> Finish)

4. Dodanie do projektu**nowej klasy zawierającej metodę main**

 File/New/Class -> wybrać nazwę klasy i zaznaczyć pole dodające metodę main -> Finish)

##### **Zadanie 2 - Aplikacja konsoli w eclipse**

W środowisku Eclipse utworzyć projekt Lab0, paczkę zadanie2 oraz klasę HelloFromConsoleposiadającą metodę main() która wypisze na ekranie frazę "Hello World!".

##### 

##### **Zadanie 3 - Aplikacja GUI w eclipse**

W środowisku Eclipse stwórz paczkę zadanie3 a w niej stwórz klasę o nazwie CreateGUI posiadającą metodę main(). Wewnątrz metody main() wpisz kolejno:

JFrame frame = **new** JFrame();

frame.setSize(640, 480);

frame.setVisible(**true**);

Uruchomienie takiego programu spowoduje pojawienie się pustego okna o zadanym rozmiarze.

Uwaga! Trzeba będzie również dodać następującą linijkę:

import javax.swing.JFrame;

Można to zrobić, używając podpowiedzi środowiska Eclipse.

Uwaga 2! Java 17, z jakiej korzystamy w naszym Eclipse, ładuje zależności w postaci tzw. modułów (ang. **modules**), czyli zestawów paczek. Żeby kompilator potrafił zaimportować pakiet JFrame, potrzeba podać odpowiedni moduł w Lab0/module-info.java:

requires java.desktop;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

**Laboratorium nr 1**

**A. Wstęp.**

Pamiętacie najprostszy kod wypisujący napis na ekran w C++? Tak wygląda to w Javie:

|  |  |
| --- | --- |
| **Hello World: C++** | **Hello World:  Java** |
| #include <iostream>    **using** **namespace** std;    **int** main(**int** argc, **char**\*\* argv)  {  cout<<"Hello World!";  **return** 0;  } | **class** Hello  {  **public** **static** void main(**String**[ ] args)  {  **System**.out.println("Hello World!");  }  } |

Podstawowe pętle i instrukcje warunkowe są identyczne w obu językach:

|  |  |
| --- | --- |
| **Kod działający zarówno w C++ jak i w Javie** | |
| int foo=1, bar=3;  if (foo == 1) {  foo=2;  }  else {  foo=3;  }  for (int ii = 0; ii < bar; ii++){  bar += 5 % 2;  } | |
| **Kompilacja: C++** | **Kompilacja: Java** |
| g++ [lista plików do skompilowania] | javac [lista plików do skompilowania] |

|  |  |
| --- | --- |
| **Wynik kompilacji: C++** | **Wynik kompilacji: Java** |
| program: a.out | plik .class: Hello.class |

W Javie po poprawnej kompilacji tworzone są pliki .class, które mogą być uruchomione w dowolnym systemie operacyjnym posiadającym Wirtualną Maszynę Javy (JAVA VM).

**Uwaga**: Nazwy plików z kodem źródłowym Javy NIE SĄ dowolne:

1. muszą mieć rozszerzenie **.java**.
2. w przypadku deklarowania klasy publicznej, nazwa pliku musi być identyczna z nazwą klasy (wielkość liter ma znaczenie)

Przykładowo, klasa:

public class Example {...} musi być zadeklarowana w pliku Example.java i w wyniku kompilacji utworzy  Example.class

|  |  |
| --- | --- |
| **Uruchomienie: C++** | **Uruchomienie: Java** |
| ./a.out | java [nazwa pliku class] |

|  |  |
| --- | --- |
| **Uruchomienie z argumentami: C++** | **Uruchomienie z argumentami: Java** |
| ./a.out arg1 arg2 | java [nazwa pliku class] arg1 arg2 |

**Tablice**

A oto przykład deklaracji tablic w Javie (w porównaniu z C++).

|  |  |
| --- | --- |
| **Tablice: C++** | **Tablice:  Java** |
| **double** myList1[10];  **double**\* myList2; //deklaracja  myList = **new** **double**[10]; //utworzenie tablicy  **double**\* myList = **new** **double**[10]; //dwa w jednym  **double** total = 0;  **for** (**int** i = 0; i < 10; i++) {  total += myList[i];  }  **delete** [] myList;  **delete** [] myList2; | double[] myList1 = new double[10];  double[] myList2; //deklaracja  myList = new double[10]; //utworzenie tablicy  double[] myList = new double[10]; //dwa w jednym  double total = 0;  for (int i = 0; i < myList.length; i++) {  total += myList[i];  } |

Co należy zauważyć:

* W Javie nie występują "tablice statyczne". Tablice zawsze deklarujemy przez **new**.
* Nie ma **delete**! W Javie **Garbage Collector**czyści pamięć za nas.
* Pole **length**tablicy przechowuje jej długość.
* Odczyt oraz zapis do poszczególnych komórek tablicy jest taki sam w obu językach.

**B. Eclipse**

Na zajęciach nie będziemy jednak pisać w notatniku, tylko korzystać ze środowiska **Eclipse**. Eclipse to bardzo popularne **zintegrowane środowisko programistyczne** (ang. **IDE**), czyli zespół aplikacji służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Pozwala na instalację dużej ilości różnych wtyczek, dopasowanych do potrzeb programisty.

Podstawowe informacje które mogą się przydać dzisiaj na zajęciach:

1. Po uruchomieniu zostaniesz zapytany o **workspace** - katalog, w którym będą zapisywane nowe projekty

2. Tworzenie**nowego projektu**w Eclipse

Wystarczy wybrać z menu: (File/New/Project... -> Java Project -> wybrać nazwę projektu -> Finish)

3. Dodanie do projektu **nowej klasy**

 File/New/Class -> wybrać nazwę klasy -> Finish)

4. Dodanie do projektu**nowej klasy zawierającej metodę main**

 File/New/Class -> wybrać nazwę klasy i zaznaczyć pole dodające metodę main -> Finish)

5. Uruchamianie programu **z argumentem** w środowisku Eclipse: Run -> Run Configurations -> wybrać zakładkę arguments -> wpisać argumenty programu

**Paczki java (package i import)**

W dużych środowiskach problemem może być zderzenie nazw pomiędzy różnymi klasami należącymi do dwóch projektów które trzeba połączyć.

Wyobraźmy sobie taki scenariusz:  sprzedawca mebli posiadający napisany w Javie system sprzedażowo-magazynowy chce rozwinąć się w kierunku sprzedaży detalicznej i postanowił stworzyć sklep internetowy i w tym celu zakupił gotowy CMS napisany w Javie oparty o Javę EE. Niestety, jego istniejący system sprzedażowy zawiera klasę o nazwie Table.java która reprezentuje różne stoły jakie opisywana firma sprzedaje. CMS również operuje klasą Table.java, która reprezentuje tabelkę HTML. Bez operowania przestrzeniami nazw (w javie określa się je *paczkami*) nie da się tych dwóch projektów połączyć, bo kompilator nie wiedziałby do której z klas odnosi się np. wywołanie:

Table table = new Table();

**Jak umieścić klasę w paczce?**  
Jeśli chcemy umieścić naszą klasę w paczce o nazwie foo.bar musimy dodać dyrektywę  package foo.bar; jako pierwszą linię w treści klasy.

Wszystkie klasy należące do tej paczki znajdują się w katalogu foo/bar

* Jeśli mówimy o kodzie źródłowym, to znajdują się one w tym katalogu wewnątrz katalogu ze źródłami projektu.
* Jeśli mówimy o skompilowanym kodzie, to znajdują się one w katalogu foo/bar katalogu głównego w pliku .jar lub katalogu dodanego do classpath

**Import klas z paczki**

By odnieść się do klasy Baz z paczki foo.bar możemy podać jej pełną (kwalifikowaną) nazwę:

|  |  |
| --- | --- |
|  | foo.bar.Baz baz = new foo.bar.Baz(); |

Operowanie pełnymi nazwami jest niewygodne, ale czasem jest konieczne.

Przykładowo, gdy operujemy kilkoma klasami o tej samej nazwie (w przykładzie klasy Baz z paczek foo i bar).

|  |  |
| --- | --- |
|  | foo.Baz baz = new foo.Baz();  bar.Baz baz = new bar.Baz(); |

By skrócić ilość pisania, możemy użyć dyrektywy  import, która pozwala ominąć nazwę paczki. Proszę pamiętać że dyrektywa import znaczy tak naprawdę: "Drogi kompilatorze, wiedz, że od tej pory, gdy napiszę nazwę klasy Baz, chodzi mi o klasę Baz z paczki foo.bar:

|  |  |
| --- | --- |
|  | import foo.bar.Baz; |

**Związek między klasami w paczce**

Klasy w jednej paczce widzą się nawzajem, tj. nie potrzeba ich importować.

Dodatkowo klasy z jednej paczki mają ułatwiony dostęp do swoich atrybutów. Wyobraźmy sobie taką klasę:

**package** foo.bar;

**class** Foo{

**private** int priv;

int pack;

**protected** int prot;

**public** int pub;

**private** void doNothingPrivate(){};

void doNothingPack(){};

**protected** void doNothingProt(){};

**public** void doNothing(){};

}

Klasy z paczki foo.bar*nie mają dostępu* tylko do atrybutu priv i metody doNothingPrivate.

**Zadanie A - Hello i pętla. (1 pkt)**

Utworzyć nowy projekt, a następnie utworzyć w nim plik tekstowy (File → New → File) o nazwie **Hello.java.**Następnie należy napisać w jego wnętrzu program wypisujący na ekranie słowa "Witaj!" a pod spodem liczby od 1 do [pierwszego\_argumentu\_programu]. Klasa powinna znajdować się w paczce **pojava.lab1.zadA**. Następnie należy skompilować go i uruchomić.

Wskazówki:

1. Najpierw napisz program, który wyświetla liczby od 1 do X, gdzie X = 20.

          2. Zmień X na w ten sposób, by X było argumentem przekazywanym z zewnątrz do metody main(String[] args).

* Należy zauważyć, że argumenty przekazywane są do funkcji main w postaci tablicy obiektów klasy String. Czyli argumenty wywołania programu można znaleźć pisząc args[1], args[2], itp...
* Zamiana String na int:
* String myString = "1234";

int myInt = Integer.parseInt(myString);

* Program powinien wyświetlać informację o błędzie jeśli użytkownik nie poda argumentu. Najłatwiej sprawdzić, czy długość tablicy args jest równa 0.

**Zadanie B - Tablica String. (1.5 pkt)**

Używając środowiska Eclipse napisz program, który utworzy tablicę na 4 ciągi znaków (String), wypełni ją słowami (cztery pierwsze parametry wywołania programu) po czym wypisze na ekranie. Klasa powinna znajdować się w paczce **pojava.lab1.zadB**.

**C. Klasy i Dziedziczenie**

Każdy pisany w Javie kod znajduje się w obrębie jakiejś klasy. Żeby stworzyć obiekt klasy i wywołać jego metody należy użyć operatora **new:**

KlasaPrzykladowa kp = new KlasaPrzykladowa();

**Pamiętaj:** obiekty w Javie zawsze tworzone są przy pomocy operatora new po którym następuje konstruktor (czyli nazwa klasy wraz z ewentualnymi argumentami).

Dziedziczenie jest znanym państwu z C++sa mechanizmem współdzielenia funkcjonalności pomiędzy klasami:

|  |  |
| --- | --- |
| **Dziedziczenie: C++** | **Dziedziczenie:  Java** |
| **class** Bazowa  {  **public**:  Bazowa();  };  **class** Pochodna : **public** Bazowa  {  }; | **class** Bazowa  {  **public** Bazowa();  };  **class** Pochodna **extends** Bazowa  {  }; |

Co należy zauważyć:

* Słowo kluczowe **extends**mówi o dziedziczeniu z klasy nadrzędnej
* Nie ma już dwukropków po modyfikatorach dostępu (public / private itp.), każda funkcja składowa klasy i konstruktor posiada własny modyfikator dostępu.
* Domyślnym modyfikatorem nie jest już "private" jak w C++, lecz "package protected", czyli dana funkcja / klasa / pole jest dostępne z każdego miejsca danej paczki.

**Zadanie C - Dziedziczenie (2.5 pkt)**

Proszę stworzyć **klasę Auto**, która zawiera:

* pole: **float[] przebieg**; - to tablica przechowująca informację o ilości przejechanych kilometrów w kolejnych miesiącach, jej rozmiar to 12; Tablica jest inicjalizowana losowymi wartościami**w konstruktorze.**
* metodę: **float srPrzebieg()**; - oblicza średni przebieg dla samochodu.
* nie posiada metody main

Proszę stworzyć klasę Taxi, która dziedziczy po klasie Auto i zawiera:

* pole: **float[] zarobki;** - to tablica przechowująca informację o zarobkach taksówkarza kolejnych miesiącach, jej rozmiar to 12; Tablica jest inicjalizowana losowymi wartościami **w konstruktorze.**
* metodę: **float srZarobki();** - oblicza średnie zarobki dla taksówki.
* posiada metodę **main** w której proszę stworzyć obiekt klasy Taxi i wyświetlić na ekranie średni przebieg i średnie zarobki.

Wskazówka:

* każdą klasę należy zadeklarować w oddzielnym pliku (o nazwie danej klasy)
* konstruktory w Javie deklaruje się tak samo jak w C++,
* dziedziczenie również deklaruje się tak samo jak w C++, należy jedynie pamiętać o słowie **extends** zamiast dwukropka
* funkcję losującą liczby zmiennoprzecinkowe z danego zakresu (przy wypełnianiu tablic przebieg, zarobki) należy samodzielnie znaleźć w internecie

Poniżej kod dla dwóch przykładowych klas: bazowej i pochodnej.

**Klasa bazowa**

|  |  |
| --- | --- |
| **package** pl.wyklad1.klasapokazowa;  **public** **class** KlasaPrzykladowa {  // deklaracje konstruktorow  KlasaPrzykladowa() {  x = 0;  tekst = "domyslny";  //to jest konstruktor domyslny  }  // deklaracje metod  void setX(int aX) {  x = aX;  }  int getX() {  **return** x;  }  void wypisz() {  **System**.out.println(x + " " + tekst);  }  // deklaracje pol  int x;  **String** tekst; | **Klasa pochodna**  **public** **class** KlasaPochodna **extends** KlasaPrzykladowa {  KlasaPochodna(){  x = 1;  tekst = "domyslny pochodnej";  //konstruktor domyslny  }  **public** void wypisz(){  **System**.out.println(x + " " + y + " " + tekst);  }  int y = 10;  }  **Funkcja główna main**  **public** **static** void main(String[] args) {  KlasaPochodna kp = **new** KlasaPochodna();  kp.wypisz();  } |

**Laboratorium nr 2a**

**- powtórka z zajęć poprzednich dla treningu**

Wstęp. Różnice pomiędzy C++ a Javą.

**A. Wstęp.**

Pamiętacie najprostszy kod wypisujący napis na ekran w C++? Tak wygląda to w Javie:

|  |  |
| --- | --- |
| **Hello World: C++** | **Hello World:  Java** |
| #include <iostream>    **using** **namespace** std;    **int** main(**int** argc, **char**\*\* argv)  {  cout<<"Hello World!";  **return** 0;  } | **class** Hello  {  **public** **static** void main(**String**[ ] args)  {  **System**.out.println("Hello World!");  }  } |

Podstawowe pętle i instrukcje warunkowe są identyczne w obu językach:

|  |
| --- |
| **Kod działający zarówno w C++ jak i w Javie** |
| int foo=1, bar=3;  if (foo == 1) {  foo=2;  }  else {  foo=3;  }  for (int ii = 0; ii < bar; ii++){  bar += 5 % 2;  } |

|  |  |
| --- | --- |
| **Kompilacja: C++** | **Kompilacja: Java** |
| g++ [lista plików do skompilowania] | javac [lista plików do skompilowania] |

|  |  |
| --- | --- |
| **Wynik kompilacji: C++** | **Wynik kompilacji: Java** |
| program: a.out | plik .class: Hello.class |

W Javie po poprawnej kompilacji tworzone są pliki .class, które mogą być uruchomione w dowolnym systemie operacyjnym posiadającym Wirtualną Maszynę Javy (JAVA VM).

**Uwaga**: Nazwy plików z kodem źródłowym Javy NIE SĄ dowolne:

1. muszą mieć rozszerzenie **.java**.
2. w przypadku deklarowania klasy publicznej, nazwa pliku musi być identyczna z nazwą klasy (wielkość liter ma znaczenie)

Przykładowo, klasa:

public class Example {...} musi być zadeklarowana w pliku Example.java i w wyniku kompilacji utworzy  Example.class

|  |  |
| --- | --- |
| **Uruchomienie: C++** | **Uruchomienie: Java** |
| ./a.out | java [nazwa pliku class] |

|  |  |
| --- | --- |
| **Uruchomienie z argumentami: C++** | **Uruchomienie z argumentami: Java** |
| ./a.out arg1 arg2 | java [nazwa pliku class] arg1 arg2 |

**Tablice**

A oto przykład deklaracji tablic w Javie (w porównaniu z C++).

|  |  |
| --- | --- |
| **Tablice: C++** | **Tablice:  Java** |
| **double** myList1[10];  **double**\* myList2; //deklaracja  myList = **new** **double**[10]; //utworzenie tablicy  **double**\* myList = **new** **double**[10]; //dwa w jednym  **double** total = 0;  **for** (**int** i = 0; i < 10; i++) {  total += myList[i];  }  **delete** [] myList;  **delete** [] myList2; | double[] myList1 = new double[10];  double[] myList2; //deklaracja  myList = new double[10]; //utworzenie tablicy  double[] myList = new double[10]; //dwa w jednym  double total = 0;  for (int i = 0; i < myList.length; i++) {  total += myList[i];  } |

Co należy zauważyć:

* W Javie nie występują "tablice statyczne". Tablice zawsze deklarujemy przez **new**.
* Nie ma **delete**! W Javie **Garbage Collector**czyści pamięć za nas.
* Pole **length**tablicy przechowuje jej długość.
* Odczyt oraz zapis do poszczególnych komórek tablicy jest taki sam w obu językach.

**B. Eclipse**

Na zajęciach nie będziemy jednak pisać w notatniku, tylko korzystać ze środowiska **Eclipse**. Eclipse to bardzo popularne **zintegrowane środowisko programistyczne** (ang. **IDE**), czyli zespół aplikacji służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Pozwala na instalację dużej ilości różnych wtyczek, dopasowanych do potrzeb programisty. Więcej informacji: [tutaj](https://eclipseide.org/).

Podstawowe informacje które mogą się przydać dzisiaj na zajęciach:

1. Po uruchomieniu zostaniesz zapytany o **workspace** - katalog, w którym będą zapisywane nowe projekty

2. Tworzenie**nowego projektu**w Eclipse

Wystarczy wybrać z menu: (File/New/Project... -> Java Project -> wybrać nazwę projektu -> Finish)

3. Dodanie do projektu **nowej klasy**

 File/New/Class -> wybrać nazwę klasy -> Finish)

4. Dodanie do projektu**nowej klasy zawierającej metodę main**

 File/New/Class -> wybrać nazwę klasy i zaznaczyć pole dodające metodę main -> Finish)

5. Uruchamianie programu **z argumentem** w środowisku Eclipse: Run -> Run Configurations -> wybrać zakładkę arguments -> wpisać argumenty programu

**Paczki java (package i import)**

W dużych środowiskach problemem może być zderzenie nazw pomiędzy różnymi klasami należącymi do dwóch projektów które trzeba połączyć.

Wyobraźmy sobie taki scenariusz:  sprzedawca mebli posiadający napisany w Javie system sprzedażowo-magazynowy chce rozwinąć się w kierunku sprzedaży detalicznej i postanowił stworzyć sklep internetowy i w tym celu zakupił gotowy CMS napisany w Javie oparty o Javę EE. Niestety, jego istniejący system sprzedażowy zawiera klasę o nazwie Table.java która reprezentuje różne stoły jakie opisywana firma sprzedaje. CMS również operuje klasą Table.java, która reprezentuje tabelkę HTML. Bez operowania przestrzeniami nazw (w javie określa się je *paczkami*) nie da się tych dwóch projektów połączyć, bo kompilator nie wiedziałby do której z klas odnosi się np. wywołanie:

Table table = new Table();

**Jak umieścić klasę w paczce?**  
Jeśli chcemy umieścić naszą klasę w paczce o nazwie foo.bar musimy dodać dyrektywę  package foo.bar; jako pierwszą linię w treści klasy.

Wszystkie klasy należące do tej paczki znajdują się w katalogu foo/bar

* Jeśli mówimy o kodzie źródłowym, to znajdują się one w tym katalogu wewnątrz katalogu ze źródłami projektu.
* Jeśli mówimy o skompilowanym kodzie, to znajdują się one w katalogu foo/bar katalogu głównego w pliku .jar lub katalogu dodanego do classpath

**Import klas z paczki**

By odnieść się do klasy Baz z paczki foo.bar możemy podać jej pełną (kwalifikowaną) nazwę:

|  |  |
| --- | --- |
|  | foo.bar.Baz baz = new foo.bar.Baz(); |

Operowanie pełnymi nazwami jest niewygodne, ale czasem jest konieczne.

Przykładowo, gdy operujemy kilkoma klasami o tej samej nazwie (w przykładzie klasy Baz z paczek foo i bar).

|  |  |
| --- | --- |
|  | foo.Baz baz = new foo.Baz();  bar.Baz baz = new bar.Baz(); |

By skrócić ilość pisania, możemy użyć dyrektywy  import, która pozwala ominąć nazwę paczki. Proszę pamiętać że dyrektywa import znaczy tak naprawdę: "Drogi kompilatorze, wiedz, że od tej pory, gdy napiszę nazwę klasy Baz, chodzi mi o klasę Baz z paczki foo.bar:

|  |  |
| --- | --- |
|  | import foo.bar.Baz; |

**Związek między klasami w paczce**

Klasy w jednej paczce widzą się nawzajem, tj. nie potrzeba ich importować.

Dodatkowo klasy z jednej paczki mają ułatwiony dostęp do swoich atrybutów. Wyobraźmy sobie taką klasę:

**package** foo.bar;

**class** Foo{

**private** int priv;

int pack;

**protected** int prot;

**public** int pub;

**private** void doNothingPrivate(){};

void doNothingPack(){};

**protected** void doNothingProt(){};

**public** void doNothing(){};

}

Klasy z paczki foo.bar*nie mają dostępu* tylko do atrybutu priv i metody doNothingPrivate.

**Zadanie A - Hello i pętla. (1 pkt)**

Utworzyć nowy projekt, a następnie utworzyć w nim plik tekstowy (File → New → File) o nazwie **Hello.java.**Następnie należy napisać w jego wnętrzu program wypisujący na ekranie słowa "Witaj!" a pod spodem liczby od 1 do [pierwszego\_argumentu\_programu]. Klasa powinna znajdować się w paczce **pojava.lab1.zadA**. Następnie należy skompilować go i uruchomić.

Wskazówki:

1. Najpierw napisz program, który wyświetla liczby od 1 do X, gdzie X = 20.

          2. Zmień X na w ten sposób, by X było argumentem przekazywanym z zewnątrz do metody main(String[] args).

* Należy zauważyć, że argumenty przekazywane są do funkcji main w postaci tablicy obiektów klasy String. Czyli argumenty wywołania programu można znaleźć pisząc args[1], args[2], itp...
* Zamiana String na int:
* String myString = "1234";

int myInt = Integer.parseInt(myString);

* Program powinien wyświetlać informację o błędzie jeśli użytkownik nie poda argumentu. Najłatwiej sprawdzić, czy długość tablicy args jest równa 0.

**Zadanie B - Tablica String. (1.5 pkt)**

Używając środowiska Eclipse napisz program, który utworzy tablicę na 4 ciągi znaków (String), wypełni ją słowami (cztery pierwsze parametry wywołania programu) po czym wypisze na ekranie. Klasa powinna znajdować się w paczce **pojava.lab1.zadB**.

**C. Klasy i Dziedziczenie**

Każdy pisany w Javie kod znajduje się w obrębie jakiejś klasy. Żeby stworzyć obiekt klasy i wywołać jego metody należy użyć operatora **new:**

KlasaPrzykladowa kp = new KlasaPrzykladowa();

**Pamiętaj:** obiekty w Javie zawsze tworzone są przy pomocy operatora new po którym następuje konstruktor (czyli nazwa klasy wraz z ewentualnymi argumentami).

Dziedziczenie jest znanym państwu z C++sa mechanizmem współdzielenia funkcjonalności pomiędzy klasami:

|  |  |
| --- | --- |
| **Dziedziczenie: C++** | **Dziedziczenie:  Java** |
| **class** Bazowa  {  **public**:  Bazowa();  };  **class** Pochodna : **public** Bazowa  {  }; | **class** Bazowa  {  **public** Bazowa();  };  **class** Pochodna **extends** Bazowa  {  }; |

Co należy zauważyć:

* Słowo kluczowe **extends**mówi o dziedziczeniu z klasy nadrzędnej
* Nie ma już dwukropków po modyfikatorach dostępu (public / private itp.), każda funkcja składowa klasy i konstruktor posiada własny modyfikator dostępu.
* Domyślnym modyfikatorem nie jest już "private" jak w C++, lecz "package protected", czyli dana funkcja / klasa / pole jest dostępne z każdego miejsca danej paczki.

**Zadanie C - Dziedziczenie (2.5 pkt)**

Proszę stworzyć **klasę Auto**, która zawiera:

* pole: **float[] przebieg**; - to tablica przechowująca informację o ilości przejechanych kilometrów w kolejnych miesiącach, jej rozmiar to 12; Tablica jest inicjalizowana losowymi wartościami**w konstruktorze.**
* metodę: **float srPrzebieg()**; - oblicza średni przebieg dla samochodu.
* nie posiada metody main

Proszę stworzyć klasę Taxi, która dziedziczy po klasie Auto i zawiera:

* pole: **float[] zarobki;** - to tablica przechowująca informację o zarobkach taksówkarza kolejnych miesiącach, jej rozmiar to 12; Tablica jest inicjalizowana losowymi wartościami **w konstruktorze.**
* metodę: **float srZarobki();** - oblicza średnie zarobki dla taksówki.
* posiada metodę **main** w której proszę stworzyć obiekt klasy Taxi i wyświetlić na ekranie średni przebieg i średnie zarobki.

Wskazówka:

* każdą klasę należy zadeklarować w oddzielnym pliku (o nazwie danej klasy)
* konstruktory w Javie deklaruje się tak samo jak w C++,
* dziedziczenie również deklaruje się tak samo jak w C++, należy jedynie pamiętać o słowie **extends** zamiast dwukropka
* funkcję losującą liczby zmiennoprzecinkowe z danego zakresu (przy wypełnianiu tablic przebieg, zarobki) należy samodzielnie znaleźć w internecie

Poniżej kod dla dwóch przykładowych klas: bazowej i pochodnej.

**Klasa bazowa**

|  |  |
| --- | --- |
| **package** pl.wyklad1.klasapokazowa;  **public** **class** KlasaPrzykladowa {  // deklaracje konstruktorow  KlasaPrzykladowa() {  x = 0;  tekst = "domyslny";  //to jest konstruktor domyslny  }  // deklaracje metod  void setX(int aX) {  x = aX;  }  int getX() {  **return** x;  }  void wypisz() {  **System**.out.println(x + " " + tekst);  }  // deklaracje pol  int x;  **String** tekst; | **Klasa pochodna**  **public** **class** KlasaPochodna **extends** KlasaPrzykladowa {  KlasaPochodna(){  x = 1;  tekst = "domyslny pochodnej";  //konstruktor domyslny  }  **public** void wypisz(){  **System**.out.println(x + " " + y + " " + tekst);  }  int y = 10;  }  **Funkcja główna main**  **public** **static** void main(String[] args) {  KlasaPochodna kp = **new** KlasaPochodna();  kp.wypisz();  } |

**Laboratorium nr 2b**

Podstawy tworzenia GUI - okienka, komponenty, obsługa przycisku.

**Graficzny interfejs użytkownika czyli GUI (Graphical User Interface)**

Jedną z charakterystycznych cech języka Java jest możliwość łatwego tworzenia graficznego interfejsu użytkownika (GUI). Wykorzystuje się do tego dwa podstawowe pakiety: java.awt i javax.swing.

Podstawowym obiektem, z którego zbudowane jest GUI jest okno, w którym umieszczone są wszystkie pozostałe elementy.

W pakiecie javax.swing klasą implementującą okno jest JFrame (<https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html>)

**Przykład 1. Pierwsze okno**

W środowisku Eclipse stwórz projekt pojava i dodaj do niego paczkę lab2 a w niej stwórz klasę o nazwie Example1 posiadającą metodę main(). Wewnątrz metody main() wpisz kolejno:

JFrame frame = **new** JFrame();

frame.setSize(640, 480);

frame.setVisible(**true**);

Uruchomienie takiego programu spowoduje pojawienie się pustego okna o zadanym rozmiarze, jednak kliknięcie przycisku [X] nie spowoduje zatrzymania programu. Będzie to tematem pierwszego zadania.

**Zadanie A - Dziedziczenie po JFrame. (1 pkt)**

W środowisku Eclipce stwórz projekt pojava i dodaj do niego paczkę lab2, a w niej stwórz klasę o nazwie CloseableFrame, dziedziczącą po JFrame, zawierającą metodę main() oraz definicje konstruktorów z klasy JFrame. Poprawnie utworzona klasa powinna wyglądać tak:

**public** **class** CloseableFrame **extends** JFrame {

**public** CloseableFrame() **throws** HeadlessException {

}

**public** CloseableFrame(GraphicsConfiguration gc) {

**super**(gc);

}

**public** CloseableFrame(**String** title) **throws** HeadlessException {

**super**(title);

}

**public** CloseableFrame(**String** title, GraphicsConfiguration gc) {

**super**(title, gc);

}

**public** **static** **void** main(**String**[] args) {

}

}

We wszystkich konstruktorach klasy CloseableFrame ustaw domyślny rozmiar okna jako (640,480) oraz dodaj instrukcję, która spowoduje zwolnienie zasobów i zatrzymanie programu po zamknięciu okna:

**this**.setSize(640,480);

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

Wewnątrz metody main() stwórz obiekt klasy CloseableFrame (korzystając z bezargumentowego konstruktora) i uczyń go widocznym.

Metoda super() widoczna wewnątrz konstruktorów służy do wywołania konstruktora klasy nadrzędnej (w tym wypadku JFrame). Znajduje się ono zawsze na początku konstruktora, dzięki czemu można po nim dodać dalszy kod, który sprawi, że klasa CloseableFrame będzie rozszerzeniem klasy JFrame.

**Komponenty i layout managery**

Samo okienko jest zaledwie ramką, w której umieszcza się kolejne komponenty, takie jak przyciski, etykiety, pola, listy rozwijane itp. Wymienione elementy można umieszczać bezpośrednio w oknie lub w specjalnym kontenerze implementowanym przez klasę JPanel z pakietu javax.swing.

**Przykład 2. Umieszczanie komponentów bezpośrednio w oknie**

W paczce lab2 stwórz klasę o nazwie Example2 posiadającą metodę main(). Wewnątrz metody main() wpisz kolejno:

CloseableFrame frame = **new** CloseableFrame();

JButton button1 = **new** JButton("Przycisk 1");

frame.add(button1);

JButton button2 = **new** JButton("Przycisk 2");

frame.add(button2);

frame.setVisible(**true**);

Po uruchomieniu programu powinno być widoczne okno z jednym dużym przyciskiem pokrywającym całe okno.  Dzieje się tak ponieważ do pozycjonowania obiektów w oknie używany jest tzw.**layout manager**, a domyślny **layout manager** dla klasy JFrame nazywa się BorderLayout, który przy najprostszym wywołaniu metody add(Component c) umieszcza komponent w centrum ramki i rozciąga go na całe okno. Dodanie drugiego przycisku w ten sam sposób spowoduje zasłonięcie pierwszego. BorderLayout wyróżnia 5 obszarów okna: górny (**PAGE\_START**), dolny (**PAGE\_END**), lewy (**LINE\_START**), prawy (**LINE\_END**) i centralny (**CENTER**), który jest obszarem domyślnym (zobacz <https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html>). Aby komponenty nie przekrywały się, konieczne  jest przekazanie parametru do metodyadd(Component c, int index), który będzie informował, w którym obszarze ma zostać umieszczony komponent. Zmodyfikuj metodę main() w klasie Example2:

CloseableFrame frame = **new** CloseableFrame();

JButton button1 = **new** JButton("Przycisk 1");

frame.add(button1, **BorderLayout**.PAGE\_START);

JButton button2 = **new** JButton("Przycisk 2");

frame.add(button2, **BorderLayout**.PAGE\_END);

frame.setVisible(**true**);

Rodzaj rozmieszczenia komponentów można zmienić metodą setLayout(), podając jako argument jeden z dostępnych **layout managerów**. FlowLayout układa elementy jeden obok drugiego, opcjonalnie w kilku rzędach. Aby przetestować jego działanie można umieścić w oknie więcej elementów, np. etykietę z tekstem JLabel lub pole do wpisywania tekstu JTextField. Dodaj kolejne instrukcje do metody main() w klasie Example2:

JLabel label = **new** JLabel("To jest etykieta");

frame.add(label);

JTextField field = **new** JTextField("A to pole tekstowe");

frame.add(field);

frame.setLayout(**new** **FlowLayout**());

GridLayout jest bardzo wygodnym managerem, który dzieli okno na siatkę u ustalonej liczbie kolumn i wierszy. Każda komórka takiej siatki ma identyczny rozmiar.  W powyższym przykładzie zmień managera layoutu z FlowLayout na GridLayout(2,2) lub GridLayout(4,1). Prosty tutorial dotyczący layout managerów można znaleźć na stronie <https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html>.

Więcej informacji na temat JLabel i JTextField oraz pełny opis ich metod można znaleźć w dokumentacji <https://docs.oracle.com/javase/8/docs/api/>.

Uwaga

Zwrot "umieszczanie komponentów bezpośrednio w oknie" jest skrótem myślowym. Uproszczenie to wynika z tego, że komponenty dodaje się do instancji klasy JFrame metodą add(). W rzeczywistości komponenty są umieszczane w kontenerze, który jest dodawany do okna JFrame już w momencie jego inicjalizacji. Kontenerem tym jest instancja klasy Container z biblioteki java.awt (jest to klasa nadrzędna względem klasy JPanel). Aby odwołać się do tego kontenera, np. w celu zmiany koloru tła okna, należy posłużyć się metodą getContentPane().

Dodaj poniższą instrukcję do metody main() klasy Example2 (aby efekt był widoczny trzeba ustawić FlowLayout w ramce):

frame.getContentPane().setBackground(**Color**.blue);

**Przykład 3. Umieszczanie komponentów wewnątrz panelu**

W praktyce bardzo rzadko umieszcza się komponenty bezpośrednio w oknie, które w zamyśle autorów pakietu ma być tylko ramką. Właściwszą metodą jest uprzednie dodanie do okna **panelu** (JPanel) pełniącego rolę kontenera, w którym następnie umieszcza się elementy. W jednej ramce może być wiele paneli, a każdy może mieć swój własny **layout**.

W paczce lab2 stwórz klasę o nazwie Example3 posiadającą metodę main(). Wewnątrz metody main() należy umieścić instrukcje:

CloseableFrame frame = **new** CloseableFrame();

frame.setLayout(**new** GridLayout(1,2));

JPanel panel1 = **new** JPanel();

panel1.setBackground(**Color**.white);

frame.add(panel1);

JPanel panel2 = **new** JPanel();

panel2.setBackground(**Color**.black);

frame.add(panel2);

JLabel leftLabel = **new** JLabel("Lewy panel");

panel1.add(leftLabel);

JLabel rightLabel = **new** JLabel("Prawy panel");

rightLabel.setForeground(**Color**.white);

panel2.add(rightLabel);

frame.setVisible(**true**);

Wykorzystano tutaj wygodny GridLayout, który pionowo podzielił ramkę na dwie równe części. W każdej połowie okna umieszczono jeden panel, a w każdym panelu jedną etykietę. Domyślnym layoutem obiektu klasy JPanel jest FlowLayout. Więcej informacji można znaleźć w dokumentacji <https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html>.

**Przykład 4. Proste figury geometryczne wewnątrz panelu**

Nadpisując metodę paintComponent() można umieszczać na panelu proste figury geometryczne o zadanym rozmiarze i kolorze.

W paczce lab2 stwórz klasę o nazwie DrawablePanel  dziedziczącą po klasie JPanel i posiadającą metodę main(). Dodaj do nowej klasy metodę paintComponent() zdefiniowaną poniżej. Wewnątrz metody main() zadeklaruj obiekt klasy DrawablePanel i umieść go wewnątrz okna CloseableFrame.

|  |  |
| --- | --- |
| **public** void paintComponent(Graphics g) {  **super**.paintComponent(g);    g.setColor(**Color**.red);  g.fillRect(50, 50, 150, 100);    g.setColor(**Color**.blue);  g.fillOval(250, 250, 150, 150);    } | **public** **static** void main(String[] args) {    CloseableFrame frame = **new** CloseableFrame();    DrawablePanel panel = **new** DrawablePanel();  panel.setBackground(**Color**.white);  frame.add(panel);    frame.setVisible(**true**);  } |

Klasa JPanel posiada metodę paintComponent(), zatem definicja tej metody w klasie DrawablePanel **"nadpisuje"** oryginalną definicję. Schemat działania jest tutaj taki sam jak przy konstruktorach.

Jeśli jakaś metoda w klasie dziedziczącej ma rozszerzać metodę klasy oryginalnej to konieczne jest wywołanie oryginalnej metody przy użyciu instrukcji super.paintComponent(g) na początku definicji nowej metody.

W tym konkretnym przykładzie rozszerzona metoda paintComponent() rysuje na panelu dwie figury geometryczne: czerwony prostokąt i niebieskie koło.

Rysowanie odbywa się za pośrednictwem klasy [Graphics](https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html), która jest automatycznie przekazywana do metody paintComponent().

**Zadanie B - Figury w losowych kolorach. (2 pkt)**

W paczce lab2 stwórz klasę ThreeShapesPanel, dziedziczącą po klasie JPanel i posiadającą metodę main(). Dodaj do nowej klasy metodę paintComponent i zdefiniuj ją w ten sposób aby w panelu rysowane były trzy różne figury geometryczne o różnych, losowych kolorach. W tym celu odszukaj w dokumentacji klasę służącą do generowania liczb losowych.

Zmodyfikuj metodę main() w ten sposób, aby w oknie CloseableFrame umieszczone były dwa panele tej samej wielkości, jeden klasy JPanel a drugi ThreeShapesPanel (i zajmowały całą powierzchnię ramki). W panelu klasy JPanel umieść dowolne cztery komponenty (guziki, labelki, textfieldy...) ustawione jeden nad drugim. Upewnij się, że kolory figur nie zmieniają się podczas zmieniania rozmiaru okna.

**Obiekty nasłuchujące zdarzeń**

Jednym z komponentów zaprezentowanych w Przykładzie 2 był JButton. Jednak samo dodanie przycisku do panelu nie powoduje automatycznie, że będzie on działał zgodnie z wolą programisty. Kliknięcie przycisku jest pewnym zdarzeniem wywołanym przez użytkownika i aby to zdarzenie wywołało jakąkolwiek reakcję muszą być spełnione dwa warunki:

1. Program musi wiedzieć o tym, że ma oczekiwać na kliknięcie przycisku. Innymi słowy potrzebny jest obiekt nasłuchujący zdarzeń (**listener**) na danym przycisku.

2. Obiekt nasłuchujący musi mieć podane uprzednio instrukcje, które ma wykonać w przypadku kliknięcia w przycisk.

Obiektami nasłuchującymi w Javie są tzw. **listenery**. Przykładowo, aby obsłużyć zdarzenie kliknięcia na przycisk, należy podłączyć do niego instancję interfejsu ActionListener.

**Przykład 5. Obsługa przycisku**

W paczce lab2 stwórz klasę o nazwie Example5 posiadającą metodę main(). Wewnątrz metody main() należy umieścić instrukcje:

CloseableFrame frame = **new** CloseableFrame();

JPanel panel = **new** JPanel();

frame.add(panel);

JButton exitButton = **new** JButton("Zakończ");

**ActionListener** exitListener = **new** ActionListener() {

@Override

**public** **void** actionPerformed(**ActionEvent** arg0) {

**System**.exit(0);

}

};

exitButton.addActionListener(exitListener);

panel.add(exitButton);

frame.setVisible(**true**);

Interfejs ActionListener można definiować i inicjować w różny sposób i w różnych miejscach programu. Powyższy przykład prezentuje użycie tzw. **klasy anonimowej**, czyli takiej która zostaje zdefiniowana "w locie", w momencie jej inicjalizacji. Instrukcje, które mają zostać wykonane w następstwie kliknięcia na przycisk muszą być umieszczone wewnątrz metody actionPerformed(). Temat interfejsów i listenerów jest szerzej omówiony podczas Laboratorium 3.

**Zadanie C - Okno z trzema przyciskami. (2 pkt)**

W istniejącej paczce lab2 stwórz klasę o nazwie ThreeButtonFrame, dziedziczącą po JFrame, zawierającą metodę main() oraz definicje konstruktorów z klasy JFrame. W konstruktorze nowej klasy zainicjuj trzy przyciski i dodaj je do ramki ThreeButtonFrame. Każdy z przycisków powinien wykonywać inną akcję (np. zamykać program, zmieniać tytuł okna, tekst na etykiecie lub kolor jakiegoś elementu). Wewnątrz metody main() zainicjuj obiekt klasy ThreeButtonFrame i uczyń go widocznym.