Podstawy Javy

Mateusz Bereda

COMARCH

Agenda

- Wprowadzenie
  - Komponenty JDK i JRE
  - Intellij
  - Struktura aplikacji
  - Uruchamianie projektu
  - Typy proste w Java
- Programowanie obiektowe
  - Obiektowość w Java
  - Enkapsulacja
  - Dziedziczenie
  - Polimorfizm
  - Rzutowanie
  - Interfejsy i klasy abstrakcyjne
- Programowanie obiektowe
  - modyfikatory i elementy statyczne
  - Referencje
- Możliwości języka Java i biblioteki wbudowane
  - Obsługa wejścia-wyjścia
  - Wyjątki
  - Instrukcje warunkowe
  - Pętle
  - Tablice
  - Kolekcje

COMARCH

Agenda cd.

- Tworzenie aplikacji
  - Budowanie aplikacji oraz import bibliotek przy pomocy Maven
  - Testowanie aplikacji przy pomocy JUnit
- Zagadnienia dodatkowe
  - XML
  - Debugowanie aplikacji

COMARCH

Komponenty JDK i JRE

Aplikacja napisana w języku Java, aby mogła zostać uruchomiona, musi zostać skompilowana przy pomocy specjalnego narzędzia zwanego kompilatorem. Wynikiem kompilacji nie jest ciąg instrukcji procesora (jak np. w C) tylko tzw. Bytecode.

Do uruchomienia Bytecode'u wymagany jest JVM (Java Virtual Machine), który interpretuje Bytecode na instrukcje procesora.

JRE (Java Runtime Environment) składa się z JVM oraz zestawu klas i narzędzi niezbędnych do uruchomienia programów napisanych w Java.

JDK (Java Development Kit) składa się z JRE oraz narzędzi niezbędnych do implementacji i kompilacji aplikacji napisanych w Java.

COMARCH

Instalacja JDK

Aby zainstalować JDK, należy pobrać je z oficjalnego źródła Oracle - https://www.oracle.com/technetwork/java/javase/downloads/index.html

Najnowsza wersja to 22.

W przypadku systemu operacyjnego Windows należy dodać ścieżkę do katalogu bin JDK do zmiennej środowiskowej PATH.

COMARCH

Intellij

Intellij IDEA to zintegrowane środowisko programistyczne dla Java stworzone przez firmę JetBrains. Zawiera kompilator języka Java, narzędzia do refactoringu, debugowania oraz integrację z narzędziami takimi jak GIT, Maven, JUnit itp.

Intellij należy pobrać z oficjalnej strony JetBrains - https://www.jetbrains.com/idea/

Podczas tworzenia projektu w Intellij należy podać JDK, które wcześniej zainstalowaliśmy oraz lokalizację projektu.

COMARCH

Struktura aplikacji

- katalog projektu
  - .idea - pliki używane przez Intellij, ustawienia naszego projektu
  - src - kod który napiszemy :)
  - plik .iml - plik używany przez Intellij, zawiera informacje o naszym module. W przypadku kiedy aplikacja składa się z więcej niż jednego modułu, każdy z nich będzie zawierał plik .iml
- External Libraries - zawiera wszystkie biblioteki używane z projekcie

Narzędzie Maven wprowadza nowe elementy do struktury projektu

COMARCH

Klasa startowa

Każdy program napisany w Java musi posiadać klasę startową. Klasa startowa zawiera metodę main() (lub jej odpowiednik w przypadku niektórych bibliotek). Dzięki takiemu podejściu, zarówno dla programisty jak i dla JVM, jest jasne i jednoznaczne gdzie zaczyna się program.

Uruchomienie projektu w Java

Uruchomienie aplikacji w Intellij składa się z dwóch etapów. Pierwszym z nich jest proces budowania aplikacji. Drugim właściwe uruchomienie aplikacji.

Proces budowania składa się z kilku etapów m.in. sprawdzenie poprawności składniowej i semantycznej oraz kompilacji.

Dane

Wszystkie aplikacje, bez względu na to w jakim języku są napisane operują na danych. Przetwarzają je, przechowują, analizują oraz zmieniają.

Jesteśmy w stanie (a przynajmniej do tego dążymy) określić jakiego typu jest każda z posiadanych przez nas danych. Możemy jednoznacznie określić czy jest to znak, ciąg znaków czy liczba.

Język Java jest językiem silnie typowanym co oznacza że musimy “poinformować” program jakiego typu jest każda z naszych danych.

Typy proste

- byte - może przechowywać liczby całkowite z zakresu od -128 do 127 (1 bajt)
- short - może przechowywać liczby całkowite z zakresu od -32’768 do 32’767 (2 bajty)
- int - może przechowywać liczby całkowite z zakresu od -2’147’483’648 do 2’147’483’647 (4 bajty)
- long - przechowuje liczby całkowite (8 bajtów)
- float i double (4 i 8 bajtów) - przechowują liczby zmiennoprzecinkowe. Z uwagi na to że double jest typem precyzyjniejszym i we współczesnych (64 bitowych) procesorach może działać szybciej zalecane jest stosowanie właśnie tego typu.

Typy proste

- boolean - może przechowywać wartość 0 lub 1 (false, true), wartość logiczna (1 bit)
- char - może przechowywać znak alfanumeryczny.
- String - może przechowywać łańcuchy znaków. Technicznie jest to typ obiektowy jednak z uwagi na specjalne mechanizmy języka Java wspierające ten typ uznaje się go za typ prosty.

Zmienne

Zmienna jest informacją trzymaną w pamięci komputera. Za jej pośrednictwem możemy manipulować (zmieniać, zapisywać, analizować) informacjami.

Aby łatwiej było zrozumieć człowiekowi wszystko co dzieje się w programie możemy nadawać zmiennym nazwy (a nawet musimy !!). Mogą to być dowolne ciągi znaków, mogą zawierać cyfry oraz podkreślenia.

Najbardziej popularnym sposobem nazywania zmiennych w Java jest tzw. camelCase.

Zmienne mogą mieć przypisaną wartość, jednak nie muszą !!

Operowanie na zmiennych

Żeby dane w naszej aplikacji był użyteczne musimy mieć możliwość operowania na nich. Język Java daje do dyspozycji wiele operatorów na różnych typach danych.

Dla danych liczbowych:

- + (suma)
- - (różnica)
- * (iloczyn)
- / (iloraz)
- % (reszta z dzielenia)

COMARCH

Operowanie na zmiennych

Operatory porównawcze i logiczne:

- == (równość)
- != (różność)
- >= (większy lub równy)
- <= (mniejszy lub równy)
- <, > (mniejszy, większy)
- && (koniunkcja, iloczyn logiczny)
- || (alternatywa, suma logiczna)
- ! (negacja)

COMARCH

Inkrementacja i dekrementacja

Często zdarza się potrzeba zwiększenia lub zmniejszenia wartości jakiejś zmiennej o 1. Standardowy sposób:

i = i + 1; lub i = i - 1;

Inkrementacja i dekrementacja:

i++; oraz i--;

Tablice

Są to struktury danych które pozwalają na gromadzenie większej ilości danych w uporządkowanej formie. Tablice dzielimy na jednowymiarowe oraz wielowymiarowe.

Tworzenie tablicy:

typ[] nazwa_tablicy = new typ[liczba_elementów];

typ[] tablica = {wartosc1, wartosc2, wartosc3, ...};

Wyciąganie elementu tablicy:

int jakas_zmienna = nazwa_tablicy[numer_indexu];

COMARCH

Instrukcje warunkowe

Instrukcje warunkowe jak sama nazwa wskazuje służą wykonywania instrukcji pod zadanymi warunkami. Rodzaje instrukcji warunkowych w Java:

- if / if - else / if - else if / if - else if - else
- switch

COMARCH

Instrukcje warunkowe - if

Instrukcja warunkowa if wykonuje operację jeżeli dany warunek zostanie spełniony.

```
if(A) {
    B;
}
```

A - warunek (wartość logiczna), B - instrukcje do wykonania

COMARCH

Instrukcje warunkowe - if-else

Instrukcja warunkowa if - else jest rozwinięciem instrukcji if. Pozwala ona na wykonanie jakiejś instrukcji jeśli warunek jest spełniony, w przeciwnym wypadku innej.

```
if(A) {
        B;
} else {
C;
}
```

A - warunek (wartość logiczna), B - instrukcje do wykonania jeśli warunek spełniony, C - instrukcje do wykonania jeśli warunek nie został spełniony

Instrukcje warunkowe - if - else if - else

Kolejna rozbudowa instrukcji warunkowej if. Pozwala na sprawdzanie wielu warunków.

```
if(A) {

B;

} else if (C) {

D;

} else {

E;

}
```

A, C - warunek (wartość logiczna),

B - instrukcje do wykonania jeśli spełniony warunek A,

D - instrukcje do wykonania jeśli nie został spełniony warunek A ale został spełniony warunek C,

E - instrukcje do wykonania jeśli nie spełnione A i C

COMARCH

Instrukcje warunkowe - switch

Instrukcja warunkowa sprawdzająca czy wyrażenie przyjmuje określoną stałą wartość i w zależności od niej wykonuje odpowiednie instrukcje.

```
switch(A) {
        case B:
                C;
                break;
        case D:
                E;
                break;
        case F:
                G;
                break;
        default:
                H;
                break;
}
```

A - wyrażenie
B, D, F - określone wartości
C - instrukcje wykonywane jeśli A ma wartość B
E - instrukcje wykonywane jeśli A ma wartość D
G - instrukcje wykonywane jeśli A ma wartość F
H - instrukcje wykonywane jeśli A nie ma żadnej z wartości B, D lub F

COMARCH

Pętle

Jedną z podstawowych zasad pisania dobrego kodu jest unikanie powtórzeń. Pętle wykorzystuje się do wielokrotnego wykonywania tych samych instrukcji. Mogą być one skończone (posiadające warunek końcowy) lub nieskończone (kiedy warunek końcowy nigdy nie będzie spełniony). Pętle występują w większości języków programowania i działają praktycznie identycznie. Różnią się jedynie słowami kluczowymi (składnią).

Pętle

- for
- for-each
- while
- do-while

COMARCH

Pętle - for

Pętlę for zazwyczaj wykorzystujemy jeśli wiemy ile razy pewien blok kodu powinien zostać wykonany. Pętla ta posiada “licznik” który jest aktualizowany po każdym wykonaniu pętli.

```
for (A; B; C) {

    D;

}
```

A - wartość początkowa licznika, B - warunek działania pętli, C - modyfikator licznika, D - instrukcje wykonywane wielokrotnie

Pętle - for-each

Pętla for-each jest bardzo podobna do pętli for. Jedyną różnicą jest brak konieczności definiowania warunku końcowego. Pętla ta wykorzystywana jest, kiedy chcemy wykonać te same instrukcje dla każdego elementu tablicy, listy, kolekcji itp.

```
for (A : B) {

    C;

}
```

A - obiekt pobrany z tablicy lub kolekcji, B - tablica lub kolekcja (agregat danych), C - instrukcje wykonywane wielokrotnie

Pętle - while

Pętla while wykorzystywana jest, kiedy nie wiemy ile razy chcemy wykonać pewne instrukcje (np. wydajemy pieniądze dopóki je mamy, na początku nie wiem co i ile kupimy)

```
while (A) {
    B;
}
```

A - warunek wykonania, B - instrukcje wykonywane wielokrotnie

Pętle - do-while

Pętla ta różni się od pętli while tylko tym że warunek wykonania sprawdzany jest na końcu, a nie na początku jak w przypadku pętli while. Daje to nam pewność, że pętla wykona się przynajmniej jeden raz.

```
do {
    A;
} while(B)
```

A - instrukcje wykonywane wielokrotnie, B - warunek wykonania pętli

Pętle - sterowanie

Może zdarzyć się sytuacja w której chcemy przerwać działanie całej pętli lub jej konkretnego przebiegu pod określonym warunkiem. Służą do tego dwa słowa kluczowe w Języku Java:

- break - przerywa działanie całej pętli, żadna linijka kodu znajdująca się wewnątrz pętli nie zostanie już wykonana.
- continue - przerywa dany przebieg pętli (ten który aktualnie się wykonuje), pętla zachowuje się tak jakby zakończyła wewnętrzny blok kodu i rozpoczyna kolejny przebieg.

Programowanie obiektowe

Jest paradygmatem programowania, który opiera się o tworzenie aplikacji w taki sposób aby jak najlepiej odzwierciedlić rzeczywistość (problem, który program rozwiązuje) poprzez modelowanie danych w sposób zrozumiały przez człowieka.

COMARCH

Obiektowość w Java - klasy

Klasa to podstawowy element każdej aplikacji zaimplementowanej w języku Java. Jest to definicja pewnego rodzaju obiektów (nowy typ danych) które posiadają określone cechy oraz zachowanie. Każdą klasę (definicję nowego typu danych) umieszczamy w osobnym pliku (są pewne wyjątki), który musi nazywać się tak samo jak klasa i mieć rozszerzenie .java.

Klasy zawsze nazywamy wielką literą, a w przypadku wielowyrazowych nazw stosujemy camelCase.

Obiektowość w Java - klasa i obiekt

Obiekty są instancjami (konkretnymi egzemplarzami) klasy. Jeśli klasą jest Pies to instancją będzie mój pies, Twój pies i pies sąsiada. Każdy z nich jest indywidualnością. Twój pies jest grzeczny, ale pies sąsiada codziennie zjada jego kapcie. Mimo to nadal jest psem. Jednak jest to inny pies niż Twój.

Istnieje jedna klasa opisująca psa, jednak jest wiele obiektów (instancji, egzemplarzy) tej klasy.

Obiekty danej klasy tworzymy przy pomocy słowa kluczowego new. Każdy z obiektów w Java posiada swoją referencję.

Obiekty w Java mogą przyjmować wartość null. Oznacza ona brak wartości.

Obiektowość w Java - klasy

Klasy mogą zawierać:

- określone cechy, które w rzeczywistości są zmiennymi, nazywamy je polami klasy, ich ilość oraz typy są dowolne.
- określone zachowania, działania, które nazywamy metodami.
- klasy - możemy utworzyć klasę wewnątrz klasy, takie klasy nazywamy klasami wewnętrznymi

Obiektowość w Java - metody

Dla uporządkowania oraz przejrzystości i czytelności kodu program dzielimy zazwyczaj na mniejsze podprogramy. Wyróżniamy dwa typy podprogramów:

- procedury - podprogramy wykonujące jakąś czynność
- funkcje - podprogramy wykonujące jakąś czynność (np. obliczenia) i zwracające wynik

W języku Java zarówno procedury jak i funkcje nazywamy metodami.

Metody oprócz wartości zwracanej mogą przyjmować dowolną liczbę argumentów dowolnego typu.

COMARCH

Obiektowość w Java - metody

Sposób definiowania metod:

```
typ_zwracany nazwaFuncji(typ_argumentu nazwa argumentu, …) {

        kod;

}
```

Nazwa metody zawsze powinna zaczynać się z małej litery (stosujemy camelCase). Może zawierać cyfry jednak nie może zaczynać się od cyfry. Nazwa metody opisuje czynność którą metoda wykonuje np. sluchajNaSzkoleniu(). W nazwach metod nie używamy znaków specjalnych (?, %, ń, ś).

COMARCH

Obiektowość w Java - przeciążanie metod

Klasa może zawierać dwie metody o tej samej nazwie jednak nie o takiej samej definicji. Przeciążone metody posiadają tę samą nazwę, mogą (ale nie muszą) mieć inny typ zwracany oraz muszą przyjmować różne argumenty. To właśnie argumenty przekazywane do metody dają możliwość na “domyślenie się” przez Javę która z implementacji metody powinna zostać użyta.

Obiektowość w Java - konstruktor

Konstruktor w Java to specjalna metoda klasy. Jest ona uruchamiana zawsze podczas tworzenie obiektu tej klasy. Konstruktor ma zawsze taką samą nazwę jak klasa. Wewnątrz konstruktora określamy instrukcje jakie mają się wykonać podczas tworzenia nowego obiektu danej klasy. Konstruktor może przyjmować parametry a jego typem zwracanym jest zawsze obiekt klasy w której się znajduje (którą ma za zadanie skonstruować).

Obiektowość w Java - this

this jest specjalnym słowem kluczowym w języku Java. Słowo to oznacza referencję do obiektu w którym się znajduje. Może być również wykorzystywane do wołania innego konstruktora klasy.

COMARCH

Modyfikatory dostępu

Modyfikatory dostępu określają widoczność pól oraz metod klasy dla innych klas. W języku Java istnieją cztery modyfikatory:

- public - widoczność w całym projekcie
- protected - widoczność w klasach tego samego pakietu oraz dziedziczących
- none (domyślny, brak modyfikatora) - widoczność w klasach z tego samego pakietu
- private - widoczność tylko i wyłącznie wewnątrz klasy

	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
none	Y	Y	N	N
private	Y	N	N	N

Enkapsulacja

Jej inna nazwa to hermetyzacja. Jest to praktyka ukrywania pól klasy poprzez modyfikator private oraz dawanie dostępu do tych pól przez specjalne metody dostępowe nazywane getterami i setterami. Pozwala to na w pełni kontrolowany dostęp do właściwości klasy zaprojektowany przez jej twórcę. W praktyce pozwala uniknąć bardzo wielu błędów spowodowanych przez nieumiejętne korzystanie z klasy.

Dziedziczenie

Jest to mechanizm umożliwiający przejmowanie cech (pól) oraz zachowań (metod) od innych klas. Umożliwia tworzenie nowych klas na bazie już istniejących. Język Java nie umożliwia wielodziedziczenia. Każda z klas może dziedziczyć tylko i wyłącznie po jednej klasie. Dziedziczenie w Java odbywa się przez specjalne słowo kluczowe extends:

class Dog extends Pet

Oznacza to, że Pies rozszerza Zwierzę, czyli zachowuje jego funkcje i właściwości dodając przy tym nowe. Przykład:

Zwierzę potrafi chodzić i jeść. Pies dziedziczy po Zwierzęciu więc również to potrafi ale dodatkowo może jeszcze aportować (nie każde zwierzę to potrafi).

Przesłanianie metod

Jest to mechanizm języka Java pozwalający na “nadpisywanie” implementacji danej metody w klasie pochodnej.

Przykład:

Klasa zwierzę posiada metodę bawięSię(). Wszystkie zwierzęta które mamy bawią się w taki sam sposób - cieszą się i merdają ogonem (np. pies, kot, mysz). Mamy jednak nowe zwierzę - pająka. On bawi się w trochę inny sposób - nadal się cieszy ale nie merda ogonem. Możemy więc w przypadku pająka przesłonić metodę bawięSię(). Dzięki temu wszystkie nasze zwierzęta nadal bawią się “po staremu” jednak nasz nowy pająk mimo, iż jest zwierzęciem bawi się nieco inaczej.

Warunkiem przesłonięcia metody jest dokładnie taka sama definicja jak w klasie nadrzędnej (ten sam typ zwracany, argumenty oraz nazwa metody). Jedynym wyjątkiem jest modyfikator dostępu - może być on inny ale nie bardziej restrykcyjny niż w klasie nadrzędnej.

Polimorfizm

Inaczej nazywany wielopostaciowością. W języku Java oznacza możliwość traktowania obiektów różnych typów w taki sam sposób jeżeli łączy je relacja dziedziczenia. W praktyce oznacza to, iż możemy traktować zarówno ciężarówkę jak i motocykl w taki sam sposób jak każdy inny pojazd. Mimo że są to różne obiekty to cały czas są pojazdami.

Interfejsy

Interfejs jest zbiorem definicji metod (zachowań) bez ich implementacji. Każda klasa która implementuje (dziedziczy) interfejs musi implementować wszystkie metody tego interfejsu. Jako że interfejs zawiera tylko definicje metod bez ich implementacji nie możemy stworzyć obiektu na podstawie interfejsu. Możemy natomiast stworzyć obiekt klasy implementującej interfejs. Klasa może implementować wiele interfejsów jednocześnie. Definiując metody w interfejsie nie musimy podawać żadnych modyfikatorów dostępu - każda metoda domyślnie (i obowiązkowo) jest publiczna.

Klasy abstrakcyjne

Są klasami posiadającymi cechy zarówno zwykłych klas jak i interfejsów. Mogą posiadać pola i metody (jak zwykła klasa). Dodatkowo mogą posiadać metody abstrakcyjne, czyli takie które nie zawierają implementacji, a jedynie definicję (tak jak w interfejsach). Każda klasa dziedzicząca po klasie abstrakcyjnej musi implementować jej wszystkie metody abstrakcyjne. Podobnie jak w przypadku interfejsów nie można utworzyć obiektu klasy abstrakcyjnej.

Słowo kluczowe final

W języku Java słowo kluczowe final oznacza coś ostatecznego czego nie można już zmieniać.

Jeśli oznaczymy w ten sposób klasę to żadna inna klasa nie będzie mogła po niej dziedziczyć.

Jeśli oznaczymy w ten sposób metodę klasy to żadna klasa dziedzicząca nie będzie mogła przesłonić tej metody.

Oznaczenie zmiennej typu prostego oznacza, iż wartość tej zmiennej nie będzie mogła być już zmieniona.

Oznaczenie zmiennej typu obiektowego oznacza, iż referencja do obiektu nie może zostać zmieniona (sam obiekt może się zmienić).

Słowo kluczowe static

Pozwala na utworzenie bytu (pola lub metody), który nie potrzebuje obiektu aby istnieć. W praktyce oznacza to, że możemy używać metod i pól statycznych bez posiadania obiektu klasy w której się znajdują. Statyczne pole lub metoda w klasie posiada jeden egzemplarz współdzielony przez wszystkie obiekty (egzemplarze) tej klasy. Wszystkie elementy statyczne programu tworzone są podczas uruchamiania aplikacji. Można również definiować statyczne bloki kodu, które zostaną wykonane podczas uruchamiania aplikacji. Statyczne metody oraz bloki kodu mogą odwoływać się tylko i wyłącznie do statycznych pól.

Rzutowanie i konwersja danych

Jest to zmiana typu danych. Konwersja następuje w momencie kiedy JVM sam wnioskuje, że dane powinny zostać przekonwertowane na inny typ. Rzutowanie jest jawną konwersją którą programista wykonuje świadomie.

Kolekcje

Kolekcje to struktury danych służące do przechowywania zbiorów danych oraz operowania na nich. Kolekcje mogą być typowane - takie które przechowują dane tego samego typu, lub nietypowane - takie które przechowują dowolne dane. Jako że język Java jest językiem silnie typowanym nie powinniśmy używać tablic nietypowanych, ponieważ wprowadza to konieczność sprawdzania typu elementu kolekcji, który chcemy użyć.

Kolekcje

W Java wyróżniamy różne trzy główne rodzaje kolekcji:

- Listy
- Set’y
- Kolejki

Każdy rodzaj posiada kilka implementacji.

COMARCH

Listy

Najbardziej interesujące są dwa rodzaje list:

- ArrayList - przechowuje obiekty w dynamicznej tablicy, dane zapisywane znajdują się w pamięci obok siebie
- LinkedList - przechowywane obiekty “opakowane” są specjalnymi obiektami wskazującymi następnik i poprzednik, dane nie znajdują się w pamięci obok siebie

Set

Set jest zbiorem danych przechowującym unikalne wartości. Najpopularniejsze to:

- HashSet - zbiór nieuporządkowany
- TreeSet - zbiór uporządkowany

COMARCH

Mapy

Technicznie nie jest kolekcją jednak formalnie pasuje do definicji kolekcji. Potrafi przechowywać pary klucz-wartość. Każdy klucz może posiadać tylko jedną wartość. Oznacza to, że “wrzucenie” do mapy pary z kluczem który już istnieje spowoduje nadpisanie starej wartości. Najczęściej używana implementacja to HashMap.

Strumienie wejścia/wyjścia

Do komunikacji programu z użytkownikiem używa się strumieni wejścia/wyjścia. Głównym strumieniem w Java są System.out i System.in. W przypadku prostych programów System.out i System.in to konsola na której możemy zobaczyć informacje wyświetlane przez program jak i wczytać informacje wpisane w konsoli przez użytkownika. Do wczytywanie służy nam obiekt wbudowany w Java o nazwie Scanner.

Wyjątki

Czasem zdarza się, że w naszej aplikacji coś pójdzie nie tak jak zaplanowaliśmy. Java posiada mechanizm wyjątków, który jest w stanie sygnalizować nieprawidłowe sytuacje. Jako że aplikacja jest w stanie sygnalizować niepożądane sytuacje jesteśmy w stanie na nie reagować - nazywamy to obsługą wyjątków. Co więcej możemy deklarować swoje własne wyjątki. Wyjątki dzielimy na dwa rodzaje:

- Error - klasy dziedziczące po Error oznaczają błąd powodujący, że aplikacja nie może działać stabilnie lub nie może działać w ogóle. Błędy tego typu rzucane są przez JVM
- Exception - klasy dziedziczące po Exception oznaczają błędy z którymi aplikacja powinna sobie poradzić

Obsługa wyjątków

Aby obsługiwać wyjątki w Java do dyspozycji mamy blok try … catch.

```
try {
        A;
} catch (B) {
        C;
}
```

A - kod który może spowodować wyjątek, B - typ wyjątku który zamierzamy “przechwycić”, C - kod obsługi wyjątku

Przekazywanie wyjątków

Kiedy któraś z metod programu wykonuje “niebezpieczne” operacje oraz nie chcemy obsługiwać rzucanego przez nie wyjątku wewnątrz tej metody możemy przekazać wyjątek wyżej (do miejsca gdzie ta metoda zostaje wywołana). Tam nadal musi zostać on obsłużony. Służy do tego słowo kluczowe throws.

```
public void niebezpiecznaOperacja() throws StrasznyException {

....

}
```

Zapis ten możemy interpretować w taki sposób: metoda niebezpiecznaOperacja() może się nie powieść i zwrócić wyjątek typu StrasznyException.

Garbage Collector

Inaczej GC, jest komponentem JVM odpowiedzialnym za czyszczenie pamięci z nieużywanych już zmiennych i obiektów. Gdyby mechanizm ten nie istniał pamięć komputera nie byłaby zwalniana, a w efekcie po pewnym czasie nie dałoby się tworzyć nowych obiektów i zmiennych. Nie jest on oczywiście rozwiązaniem wszystkich problemów i nadal da się napisać program który zajmie całą dostępną pamięć.

XML

Jest to prosty język znaczników pozwalający na przedstawianie danych w ustrukturyzowany sposób, zrozumiały zarówno dla ludzi jak i dla maszyn. Jest on niezależny od platformy co umożliwia wymianę dokumentów (danych) pomiędzy różnymi systemami. Każdy plik XML musi posiadać dokładnie jeden element główny nazywany root. Każdy z elementów może posiadać atrybuty.

Maven

Maven jest narzędziem do zarządzania projektem, jego zależnościami (np. bibliotekami, których używamy w projekcie) oraz strukturą. Wspomaga proces budowania aplikacji oraz pozwala na wykonywanie podczas niego dodatkowych czynności poprzez stosowanie specjalnych programów zwanych plugin'ami (wtyczkami). Projekt mavenowy definiuje się poprzez stworzenie i utrzymywanie specjalnego pliku pom.xml. Właśnie w nim znajdują się wszystkie istotne elementy definiujące projekt.

JUnit

Jest narzędziem służącym do pisania powtarzalnych testów jednostkowych. Test jednostkowy to sposób testowania małej wydzielonej części programu którą testujemy w odosobnieniu. W naszym przypadku taką pojedynczą częścią może być metoda.

COMARCH

JUnit

Aby testować aplikację przy pomocy tej biblioteki należy dodać jej zależność do projektu mavenowego (pliku .pom). Każdy test jednostkowy to tak naprawdę metoda która testuje naszą jednostkę (np. metodę). Każda z takich metod musi posiadać adnotację @Test i znajdować się w klasie testującej (w katalogu test). Do sprawdzania czy testowana metoda zadziałała poprawnie używamy specjalnej klasy Assert. Daje ona możliwość sprawdzenia czy wynik działania testowanej jednostki jest zgodny z oczekiwaniami. Wiele środowisk programistycznych (w tym Intellij IDEA) posiada integrację z biblioteką JUnit.

Debugowanie aplikacji

Jest to proces wyszukiwania błędów w programie oraz ich usuwania. Większość środowisk programistycznych dla Java (w tym oczywiście Intellij IDEA) dostarczają nam wielu wygodnych narzędzi nazywanych debugerami. Narzędzia te pozwalają nam na wstrzymanie wykonywania programu w wybranym miejscu, podejrzenia wartości zmiennych w tym momencie oraz na dalsze wykonywanie programu w sposób krokowy.