Starter Kit for Tester – Java module

Wrocław, 27.04.2020,

Krzysztof Pojasek

Agenda

- Krótkie przypomnienie podstaw
- Słowa kluczowe static, final
- Enkapsulacja (hermetyzacja)
- Dziedziczenie i polimorfizm
- Interfejsy, klasa abstrakcyjna
- Struktury danych
- Instrukcja sterująca
- Obsługa wyjątków
- Enum
- Adnotacje
- Praca z obiektami typu String - krótko
- Czysty kod
- Klasa anonimowa
- Lambda
- Wbudowane interfejsy funkcyjne
- Stream

Agenda c.d

Krótkie przypomnienie

- Typy danych
- Klasa, obiekt
- Składowe klas
- Metoda main()

Krótkie przypomnienie

Typy danych

- void - nic
- short - od -32 768 do 32 767
- int - od -2 147 483 648 do 2 147 483 647
- long - od -2^63 do (2^63)-1
- float - max ok 6-7 liczb po przecinku
- double - max ok 15 cyfr po przecinku
- boolean – true/false
- char – znak
- typ obiektowy – np. String, Integer, BigDecimal, Pojazd

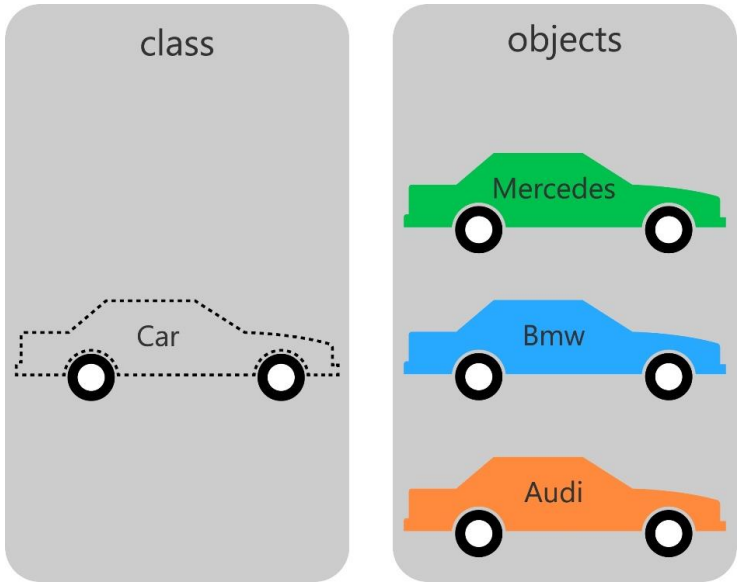
Klasa, obiekt.

Klasa

- Podstawowa komórka informacji
- Zawiera zmienne, metody
- Szablon do tworzenia obiektów
- Słowo kluczowe „**class**”

Obiekt

- Naturalna reprezentacja klasy
- Utworzenie nowego obiektu - „**new”**



Składowe klasy

- Nazwa pakietu
- Importy
- Komentarze
- Deklaracja klasy
- Konstruktor
- Deklaracje atrybutów
- Deklaracje metod

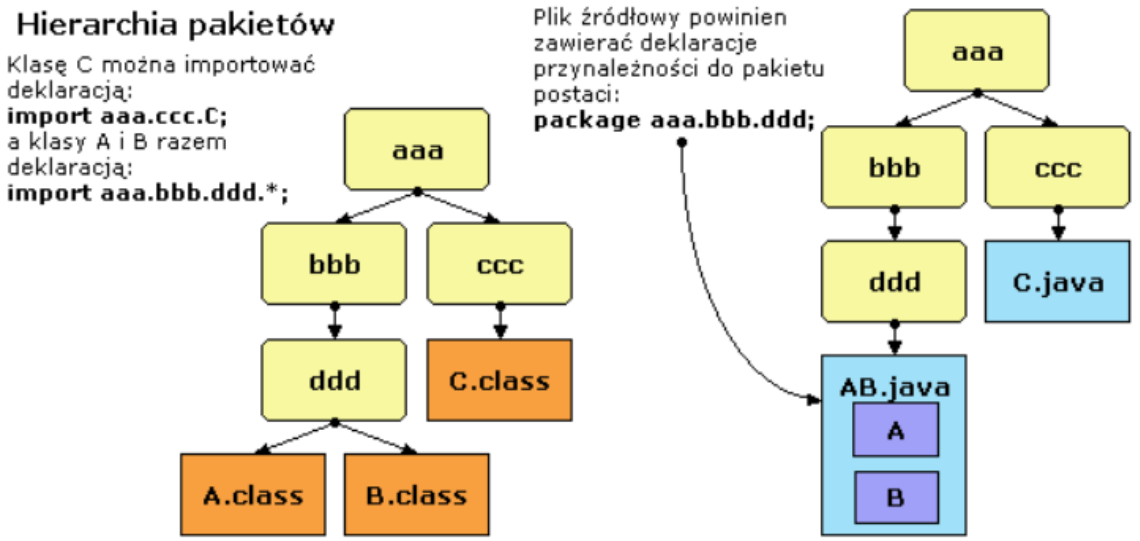
Application.java

```
1  package main.java.starterkit;
2
3  import java.util.*;
4
5  public class Application {
6      private String name = "Application"; // pole metody
7
8      //Konstruktor
9      public Application(){
10     }
11
12     private void method(String name){
13         //do nothing
14     }
15
16 }
17
```

Składowe klasy

Pakiety

- Pakiety to inaczej struktura folderów, w której przechowujemy nasz kod.
- Słowo kluczowe **„package”**



Konstruktor

- Nazwa jest identyczna z nazwą klasy, w której występują
- Nie określa zwracanego typu (nawet void)
- Brak konstruktora – konstruktor domyślny

Jak uruchomić program?

Metoda main(String [] args)

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

}
```

Słowa kluczowe – „static” i „final”

- static
  - Tworzona podczas uruchomienia programu
  - Może być klasa, metoda, pole
  - Statyczne pole/metoda to właściwość klasy nie obiektu
  - W metodzie statycznej nie możemy odwoływać się do pól i metod nie zadeklarowanych jako statyczne.

- final
  - wartość pola po zainicjowaniu nie może być zmieniona

Najlepszym przykładem wykorzystania final są stałe wykorzystywane do obliczeń.

Enkapsulacja (hermetyzacja)

Modyfikatory dostępu

Widoczność	Public	Protected	Default	Private
Do tej samej klasy	Tak	Tak	Tak	Tak
Dla każdej innej klasy z tej samej paczki	Tak	Tak	Tak	Nie
Do klasy pochodnej z tej samej paczki	Tak	Tak	Tak	Nie
Do klasy pochodnej z innej paczki	Tak	Tak	Nie	Nie
Do innej klasy, która nie jest klasą pochodną i znajduje się w innej paczce	Tak	Nie	Nie	Nie

Dziedziczenie i polimorfizm

Dziedziczenie

- mechanizm, który pozwala na rozszerzenie funkcjonalności klasy bazowej
- pozwala ograniczyć duplikację kodu
- słówko extends

```
public class Kabriolet extends Samochod{…}
```

- odwołanie się do klasy bazowej przy pomocy super()

```
public Kabriolet(String marka, String typDachu){
    super(marka);
    this.typDachu = typDachu;
}
```

Dziedziczenie i polimorfizm

Polimorfizm

- wielopostaciowość
- klasy pochodne mogą dziedziczyć stan i zachowanie klasy macierzystej
- mogą też je zmieniać

```java
class Pojazd {
    private String name;

    public Pojazd(String name){
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
class BMW extends Pojazd{
    public BMW(String name){
        super(name);
    }

    @Override
    public void setName(String name){
        name = "BMW" + name;
        super.setName(name);
    }
}
```

Interfejsy

- Zbiór metody, bez ich implementacji
- Słowo kluczowe **`implements`**
- Każda klasa implementująca interfejs musi zawierać wszystkie jego metody
- Klasa może implementować więcej niż jeden interfejs, ale dziedziczyć tylko z jednej klasy
- Nie posiadają konstruktorów
- Tworzone dla klas o podobnych właściwościach lub zachowaniach, ale nie powiązanych ze sobą (dziedziczenie)
- W odróżnieniu od klas, interfejs może rozszerzać więcej niż jeden interfejs
- Metody domyślne (default) i prywatne (private)
- W interfejsie można deklarować tylko stałe

```java
package main.java.starterkit;

public class ImplX implements InterfaceX{
    @Override
    public void print() {
        System.out.print("test");
    }
}
```

```java
package main.java.starterkit;

public interface InterfaceX extends InterfaceY, InterfaceZ{

    public void print();
}

interface InterfaceY{}
interface InterfaceZ{}
```

Klasa abstrakcyjna

- słowo kluczowe **`abstract`**
- nie można stworzyć obiektu takiej klasy,
- można ją rozszerzyć
- mogą zawierać metody bez implementacji
- może zawierać
- mogą zawierać zwykłe metody,
- wymagana implementacja metod abstrakcyjnych
- metod abstrakcyjnych nie można oznaczać jako statyczne (nie posiadają implementacji)

```java
package main.java.starterkit;

public abstract class Item {

    String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

}
```

Struktury danych

Tablice

- Tablica jest najprostszą strukturą danych.
- Tworzy się ją w następujący sposób:
- Tablice posiadają pole `length`,
  w którym przetrzymywana jest ich wielkość.
- Elementy tablicy numerowane są od 0.

```
int [] tab = new int[4];

// lub

int [] tab2 = new int [] {1,4};
```

Struktury danych

Collections

- **List -** kolekcja o określonej pozycji np. ArrayList , parametryzowana

```
List<String> lista = new ArrayList<String>();VT  //List lista = new ArrayList<String>();
        lista.add("pierwszy");
    lista.add("drugi");
    System.out.println(lista.get(1)); //wypisze „drugi"
```

Set - przechowuje obiekty bez określenia pozycji, Ten sam obiekt może występować tylko raz np. HashSet

```
Set<String> zbior = new HashSet<String>();
    zbior.add("pierwszy");
    zbior.add("drugi");
    for (String ciagZnakow : zbior) {
    System.out.println(ciagZnakow);
}
```

Struktury danych

Rodzaje list i przykładowe metody

- **Collections**
  - add(int elelment), add(int index, E element)
  - addAll(lista), addAll(int index, Collections lista)
  - clear()
  - contains(E element)
  - isEmpty()
  - remove(Object object), removeAll()
  - size()
  - toArray();

- **ArrayList**
  - get(index);
  - lista.remove(index)

- **LinkedList**
  - getFirst(), getLast()
  - get(int index) – wolniejsze
  - addFirst(E element), addLast(E element)
  - set(int index, E element)

Struktury danych

Inne kolekcje

- Queue - lista umożliwiająca implementację kolejek FIFO i FILO
- Map – nie jest to stricte kolekcja, przechowuje mapowania klucz-wartość, klucz musi być unikalny np. HashMap , TreeMap (posortowana);

```
Map<String, Integer> mapa = new HashMap<String, Integer>();
  mapa.put("pierwszy", 1);
  mapa.put("drugi", 2);
  System.out.println(map.get("pierwszy")); //wypisze "1"
```

Instrukcje sterujące

If / if-else / else-if

```
if(warunek){
   if(warunek1)
     jakaś_metoda();
   else
     jakaś_metoda1();
 }
 else if (warunek2){
   instrukcje, gdy warunek1 jest fałszywy ale
  spełniony jest warunek2
 }
```

Instrukcje sterujące

If / if-else / else-if

Jedna linijka:

```
if (warunek) jakasmetoda() else inna_metoda();
```

„Elvis operator”:

```
result = warunek ? gdy_prawda : gdy_fałsz;
```

```
return result ? 1000 : 2000;
```

```
If (result) {
     return 1000; }
else {
     return 2000;
}
```

Instrukcje warunkowe

Switch

```
switch(liczba){
case 1:
  jakieś_instrukcje_1;
  break; //zatrzymuje instrukcje switch, bez tego wykona kolejny spełniający  // warunek
case 2:
  jakieś_instrukcje_2;
  break;
...
default:
   instrukcje, gdy nie znaleziono żadnego pasującego przypadku
}
```

Instrukcje warunkowe

Pętle

for

```
for(int licznik = 0; licznik < 10; licznik++){
    System.out.println("Licznik=" + licznik);
}
```

foreach

```
for (Rower rowerItem : roweryArrayList) {
    System.out.println(rowerItem);
}
```

while / do while

```
while (roweryArrayList.size() > 0){
    roweryArrayList.remove();
}
```

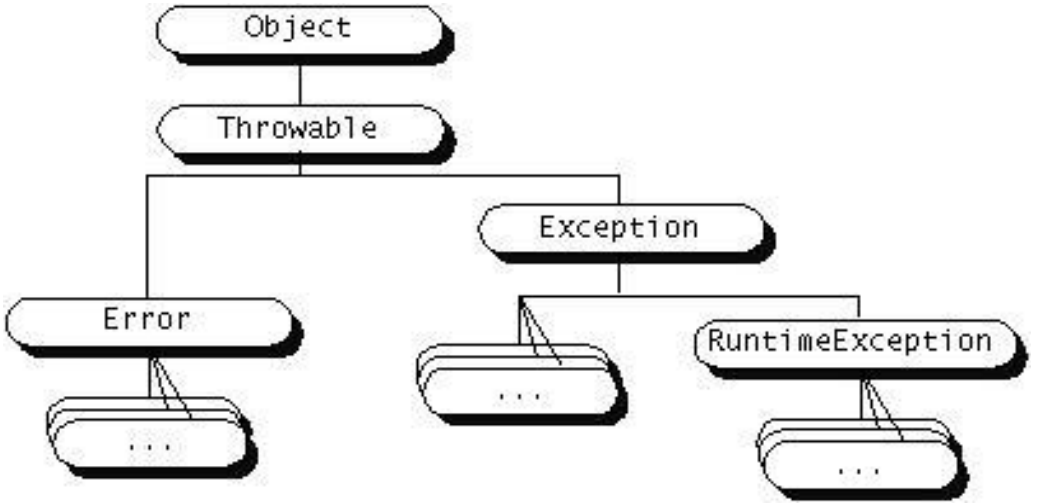
Obsługa wyjątków

Rodzaje

Errors poważne błędy w wirtualnej maszynie Javy, rzadko w programach, nie używamy

Checked *Exceptions* nie są poważne, większość programów Javy może wychwytywać i generować tego typu wyjątki. *np.* FileNotFoundException - muszą być obsłużone przez programiste

Unchecked Exceptions
dziedziczy z *Exceptions* , generowane przez aplikacj - np.: *NullPointerException.* Nie są wymagane do wychwycenia



Obsługa wyjątków

```
public void getFile(String path) throws FileNotFoundException{...}
try{
        getFile(path);
}
catch (FileNotFoundException a){
        Obsługa wyjątku a
}
catch (TypWyjątku2 b){
        Obsługa wyjątku b
}
...
finally{
        Blok instrukcji, który wykona się niezależnie, czy wyjątki wystąpią, czy  nie
}
```

Nie zostawiamy pustej klauzuli catch – zawsze tam coś powinno być!

Enum

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}

public enum KoszulkaRozmiar{
    S(45),M(47),L(49),XL(51)
}
```

- Enum – typ wyliczeniowy
- Może on posiadać atrybuty czy metody
- Domyślnie tworzony jest jako typ final
- Typ wyliczeniowy nie może określić żadnej nadklasy po której dziedziczy (nie dziedziczy po Object)

Adnotacje „@”

Służą do wygodnego ustawiania własności

```
@Test
public void Register_OK() {
(…)
}
```

```
@Override,
@Interface,
@Abstract,
```

Można tworzyć własne adnotacje

Praca z obiektami typu String

Co to jest String

String – obiekt który pozwala nam na przechowywanie sekwencji znaków

Syntax:

1. `String slowo = "slowo"; - Głównie wykorzystywany`
2. `String drugiWyraz = new String("wyraz"); - W wyjątkowych sytuacjach`

Obiekt String jest „nie mutowalny”. Wszelkie zmiany tworzą nowy obiekt.

```
String nazwaFirmy ="Apple";
String dokladnyOpis = nazwaFirmy.concat(" iPhone ");
dokladnyOpis.concat("X");
System.out.println(dokladnyOpis); // Apple iPhone
```

Praca z obiektami typu String

Konkatenacja

Syntax:

```
String s = "1";
s+="2";
s+="3";
System.out.println (s);//123
```

Różne przypadki:

```
System.out.println(1+3 );// 4
System.out.println("1"+"3" );//13
System.out.println(2+"3");//23
System.out.println(1+2+"5"); //35
```

Wybrane metody

- length()
- indexOf()
- substring()
- contains()
- replace()

Praca z obiektami typu String

StringBuilder

Typ ciągu znaków, który jest mutowalny

Przykład:

```
StringBuilder sb = new StringBuilder().append("wiekKota").append(":").append(1).append('r');
System.out.println(sb);//wiekKota:1r
```

Czysty kod

Nazewnictwo

Element języka	Zasady tworzenia nazwy	Przykłady
klasy	Nazwy klas to rzeczowniki zapisane za pomocą UpperCamelCase — pierwsza litera każdego słowa jest duża. Unikaj skrótów i akronimów, chyba, że są one powszechnie znane jak URL czy pdf czy HTML	class Raster; class ImageSprite;

metody	Nazwy metod to czasowniki zapisane za pomocą lowerCamelCase, albo wyrażenia, które zaczynają się od czasownika zapisanego małą literą, a pierwsza litera każdego kolejnego słowa jest duża.	run();
		runFast();
		getBackground();

zmienne	Zmienne lokalne, zmienne obiektu, pola klasy, są zapisywane za lowerCamelCase.	int i; String surname; float myWidth;
stałe	Nazwy stałych powinny być zapisane dużymi literami, a wyrazy w nice powinny być oddzielone podkreślikiem. Jeśli w nazwie musisz użyć liczby, to pamiętaj, by nie była ona na początku nazwy.	static final int MAX_PARTICIPANTS = 10;
pakiety	Nazwy pakietów piszemy małymi literami, zwyczajowo nazwa pakietu jest nazwą domeny, ale odwracamy kolejność poszczególnych członów.	com.example.moje.pakiety

Czysty kod

Nazewnictwo – przykłady

Tak, nie róbcie

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}
```

Tak, zdecydowanie lepiej

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if (cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Czysty kod

Metody

- Jedna metoda – jedna czynność
- Max 2 argumenty, 20 linii, 150 znaków w linii
- Odpowiednie nazewnictwo/typ (is - boolean, get - Object, set – void)

Czysty kod

Formatowanie

1. Kod podzielony na części: 1. stałe 2. pola 3. metody publiczne 4. metody prywatne
2. Zmienne jak najbliżej miejsca wykorzystania
3. Metoda wywoływana przez inną metodę powinna być zaraz pod metodą wywołującą.

Klasa anonimowa

- Definiowane w kodzie
- Jedna instancja
- Dostęp do zmiennych lokalnych oraz do zewnętrznych finalnych

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

Wyrażenia Lambda

- Tak?

```
Collections.sort(names, (String a, String b) -> {
    return b.compareTo(a);
});
```

- Albo tak?

```
Collections.sort(names, (String a, String b) -> b.compareTo(a));
```

- A może tak?

```
names.sort((a, b) -> b.compareTo(a));
```

Wyrażenia lambda

- Metoda
- Klasa anonimowa w ładniejszej postaci
- przykład: x-> x*x
- **<lista parametrów> -> <ciało wyrażenia>**
- bez argumentów : () -> „some return value”;
- `(Integer x, Long y) -> System.out.println(x * y);`
- Pomocne przy kolekcjach, strumieniach, w zapisie…
- Dostęp do pól i pól statycznych, oraz niejawnych zmiennych finalnych.

Wbudowane interfejsy funkcyjne

- Ma tylko jedną metodę abstrakcyjną
- @FunctionalInterface
- Comparator, Runnable
- Function<T, R> - apply

```
Function<Integer, Long> example = intNumber -> new Long(intNumber);
```

Wyrażenia lambda – „::”

Przypisujemy metodę do zmienne, bez wywołania

Wywołujemy gdy potrzebna do użycia

```
Object objectInstance = new Object();
IntSupplier equalsMethodOnObject = objectInstance::hashCode;
System.out.println(equalsMethodOnObject.getAsInt());
```

To to samo co :

```
Object objectInstance = new Object();
System.out.println(objectInstance.hashCode());
```

Odwołanie bez podania instancji

```
ToIntFunction equalsMethodOnObject = Object::hashCode;
Object objectInstance = new Object();
System.out.println(equalsMethodOnObject.applyAsInt(objectInstance));
```

Stream - strumienie

- Sekwencja elementów
- Operacje kończące – zwracają wynik
- Operacje pośrednie – zwracają strumień
- Tworzone na podstawie listy bądź zbioru
- Collections.stream()
- Nie modyfikuje źródła danych

Stream - filter

- Operacja pośrednia
- Przyjmuje predykat

```
intCollection
    .stream()
    .filter((s) -> s%2 == 0)
    .forEach(System.out::println);
```

Stream - map

- Operacja pośrednia
- Konwertuje na inny element

```
intCollection
    .stream()
    .map(Object::hashCode)
    .forEach(System.out::println);
```

Stream - count

- Operacja kończąca
- Zwraca liczbę elementów

```
long oddNumbers =
    intCollection
        .stream()
        .filter((s) -> s%2 == 0)
        .count();
System.out.println(oddNumbers);    // 3
```

About Capgemini

Capgemini is a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided everyday by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of 270,000 team members in nearly 50 countries. With its strong 50 year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fuelled by the fast evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering and platforms. The Group reported in 2020 global revenues of €16 billion.

Get the Future You Want | www.capgemini.com