Zadania Java

1. W Javie Stałe pole statyczne możemy zdefiniować dla:
Klasy abstrakcyjnej: **TAK**
Klasy konkretnej: **TAK**
Interfejsu: **TAK**

2. Napisz funkcję, która robi coś takiego fun(2, 10, -3) → “-3,10,2”

```
public String fun(int ... args)
{
   StringBuilder s = new StringBuilder();
   for(int i = args.length - 1; i >= 0; --i)
   {
         s.append(String.valueOf(args[i]));
         if(i == 0) continue;
         s.append(", ");
   }
   return s.toString();
}
```

3. W Javie 8 adnotację możemy postawić przed deklaracją zmiennej lokalnej w metodzie.
Prawda

```
private static void whatever()
{
    @Deprecated
    int a = 0x20103070;
}
```

Pytanie 4
Napisz statyczną metodę addNumber która:

- Zwraca wartość integer
- Ma parametr Connection conn oraz int liczba
- wstawia do tabeli o nazwie **numbers** w polu o nazwie **number** wartość przekazaną w parametrze **value**

Wykorzystaj słowa kluczowe:
`prepareStatement, executeUpdate, SQLException, close, setInt`

```
public static int addNumber(Connection conn, int value)
{
      SQLException {
            PreparedStatement ps = null;
            try {
```

```
                ps = conn.prepareStatement("INSERT INTO numbers (number)
                VALUES (?)");
                ps.setInt(1, 5);
                return ps.executeUpdate();
            } finally {
                if (ps != null)
                ps.close();
            }
      }
}
```

Pytanie 5
Napisz fragment kodu który dla zmiennej Connection o nazwie conn pobiera wszystkie pola tabeli Student.

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM Student;");
```

Pytanie 6
Jak będzie wyglądać instrukcja do uzyskania obiektu typu Connection jeśli adres to localhost:3306, nazwa użytkownika root, a hasło root:

```
DriverManager.getConnection("localhost:3306", "root", "root");
```

Pytanie 7
Obiekt typu Class dla tekstowej reprezentacji nazwy “com.mysql.jdbc.Driver” otrzymamy instrukcją:

```
Class.forName(“com.mysql.jdbc.Driver”);
```

Pytanie 8
Popraw następujący kod, tak aby mógł się skompilować:

```
int[5] arr = new int[5];
arr[5] = 4;

int[] arr = new int[5];
arr[4] = 4;
```

Pytanie 9
Napisz fragment kodu, który zawiera:

- klasę z prywatnym bezparametrowym konstruktorem i publicznym konstruktorem przyjmującym int
- uzyskaj dostęp do klasy poprzez nazwę, i wywołaj bezparametrowy konstruktor
- uzyskaj dostęp do klasy za pomocą pola “class” i wywołaj konstruktor z parametrem

```
class X
{
    private X() { System.out.println("private"); }
    public X(int x) { System.out.println("public " + x); }
```

```
}

public class Main
{
    public static void main(String[] args) throws ...
    {
        Class classByName = Class.forName("X");
        Constructor c1 = classByName.getDeclaredConstructor();
        c1.setAccessible(true);
        c1.newInstance();

        Class classByObject = X.class;
        Constructor c2 = classByObject.getConstructor(int.class);
        c2.newInstance(6);
    }
}
```

Pytanie 10
Napisz fragment kodu, który inicjalizuje Hibernate w języku Java
Configuration cfg = new Configuration().configure();

Pytanie 11
Czy `TreeSet<String>` jest podtypem `TreeSet<Object>`?
Odpowiedź: **FAŁSZ**

Pytanie 12
Jaki interfejs implementuje sterownik JDBC (Java Database Connectivity)
Odpowiedź: **java.sql.Driver**

Pytanie 13
Jaką metodą wybudza się wszystkie uśpione wątki?
Odpowiedź: **notifyAll**

Pytanie 14
W jaki sposób adnotujemy w JPA pole encji, które bedzie automatycznie numerowane? (pamiętaj o znaku@)
Odpowiedź: **@GeneratedValue(strategy = GenerationType.IDENTITY)**

Pytanie 15
W języku Java enkapsulacja oznacza, że obiekt może zachowywać się w zależności od tego, w jakim kontekście został użyty.
Odpowiedź: **FAŁSZ**

Pytanie 16
W poniższym kodzie sum.java są dokładnie 3 pluskwy(logiczne, czasu wykonania lub kompilacji).

Zaznacz, w których liniach należy poprawić kod, aby błąd nie wystąpił, a program wypisał sumę tablicy a. Jeśli nie jesteś pewien, nie zaznaczaj. Nie ma punktów ujemnych, ale za każdą pomyłkę możesz stracić 33% punktacji.

```
public class Sum {
      Sum() {
            long[] a = {5L, -10, 20, 30};
            // brak inicjalizacji zmiennej arr_sum
            long arr_sum;
            // a.length() to pole, a nie metoda powinno być a.length
            for(int i = 0; i < a.length(); i++) arr_sum += a[i];
            // nie istnieje println(String, long)
            // '+' zamiast ',' albo StringBuilder albo String.format
            System.out.println("Sum of als elements =", arr_sum);
      }
      public static void main(String args[]) {
      new Sum();
   }
}
```

Pytanie 17
Metodą domyślną w interfejsie Javy 8 poprzedzamy słowem default.
Odpowiedź: **PRAWDA**

Pytanie 18
Czy Long[] jest podtypem Object[]?
Odpowiedź: **PRAWDA**

Pytanie 19
Przyporządkuj nazwy strumieni wyjściowych do odpowiednich kategorii
Każdy użyj tylko raz, pewne mogą pozostać niewykorzystane.

Zapis sformatowanej linii do pliku tekstowego - PrintWriter
Strumień znakowy - FileWriter
Binarny zapis liczb np. float - DataOutputStream

Binarny odczyt liczb float: DataInputStream
Strumień bajtów: ?
Strumień znaków: FileWriter

Konwersja char - byte: OutputStreamWriter -- do sprawdzenia!!
Strumień obiektów, np Date: ObjectInputStream
Odczyt linii pliku tekstowego: BufferedReader

Binarny odczyt liczby float: DataInputStream
Strumień bajtów: FileInputStream
Strumień znaków: BufferedReader

InputStreamReader : byte ->char

OutputStreamWriter: char -> byte

Pytanie 20
Instrukcja super(...) w konstruktorze może się pojawić wyłącznie w pierwszej jego linii.
Odpowiedź: **PRAWDA**

Pytanie 21
Hermetyzacja w Javie może być zrealizowana poprzez prywatne składowe klasy i zastosowanie funkcji dostępowych
Odpowiedź: **PRAWDA**

Pytanie 22
Jaki napis wypisze poniższy fragment kodu?

```
String[][] arr = {{"Ala", "", "kot"},{"a", "bc", "def"}, {"g","xyz"}};
System.out.println(arr[2][1].charAt(1));
```

Odpowiedź: **y**

Pytanie 23
W Javie odwołanie do ostatniego elementu ArrayListy li to
Odpowiedź: **li.get(li.size()-1)**

```
    List<String> twojaMama = new ArrayList<>();
    twojaMama.add("Łaka");
    twojaMama.add("Maka");
    twojaMama.add("Fą");

    System.out.println(twojaMama.get(twojaMama.size()-1));
}
```

WYPISUJE "FĄ"

Pytanie 24
Dla danej tablicy (tutaj tab) utwórz strumień Javy 8 zawierający tylko długości stringów non-null i niepustych.

```java
public class Main{
    public static void main(String[]args){
            String[] tab = {"Ala", null, "ma", "", "kota","",null,"."};
            List<String> lines = Arrays.asList(tab);

            List<String> result = lines.stream()
                    .filter(line -> !"".equals(line))
                    .filter(Objects::nonNull)
                    .collect(Collectors.toList());

            result.forEach(System.out::println);
    }
}
```

25. Operacje terminalne na strumieniach Javy 8 zawsze zwracają void
Fałsz

"**Terminal** operations produces a non-stream, result such as primitive value, a collection or no value at all."
Przykłady: `toArray(), collect(), count(), reduce(), forEach(), forEachOrdered(), min(), max(), anyMatch(), allMatch(), noneMatch(), findAny(), findFirst()`

Operacje **pośrednie** (**intermediate**) z kolei zwracają strumień

26. W języku Java, jeden plik źródłowy może zawierać co najwyżej jeden publiczny interfejs.
Odpowiedź: **Prawda**

27. W Javie **Stałe pola statyczne** możemy zdefiniować dla:
Klasy abstrakcyjnej: **Tak**
Klasy konkretnej: **Tak**
Interfejsu: **Tak**

28. Java 8 wspiera wielodziedziczenie typów (dopuszcza następujący kod):

```
class C extends A, B { /* */}
```

Odpowiedź: **Fałsz**
Tylko wiele interfejsów możemy rozszerzać

29. Jaką metodą wybudza się jeden uśpiony wątek?
Odpowiedź: **notify**

31. Napisz funkcję przyjmującą jako argument **tablicę stringów** (zakładamy że argument ten jest != null oraz ma długość >0) i **zwracającą string** posiadający co najmniej 2 wystąpienia
Jeśli takiego nie ma, to niech funkcja zwraca null.
Jeśli więcej niż jeden string ma >= wystąpienia, to niech funkcja zwróci dowolny z nich.

Przykłady:
-argumentem jest tablica {“kot”,”pies”,”mysz”,”pies”}, funkcja zwraca “pies”,
-argumentem jest tablica {“kot”,”pies”,”mysz”,”koza”}, funkcja zwraca null.

Odpowiedź:

```
public String func(String[] tab) {
	LinkedList<String> list = new LinkedList<>();
	for(String s:tab) {
		if(list.contains(s)) {
			return s;
		}
		else list.add(s);
	}
	return null;
}
```

32.
Całkowite typy prymitywne Javy to: `char, int, short, long, byte`
Dodatkowo zmiennoprzecinkowe, to: `float, double`
jeszcze jest ……..: `boolean`

33. W języku java jeden plik źródłowy może zawierać co najwyżej jedną klasę publiczną.
prawda

34. W języku java jeden plik źródłowy może zawierać co najwyżej jeden publiczny interfejs.
prawda

35. Przyporządkuj nazwy strumieni **wejściowych** do odpowiednich kategorii.
Każdy użyj tylko raz, pewne mogą pozostać niewykorzystane

Binarny odczyt liczb float: **DataInputStream**
Strumień bajtów: ?
Strumień znaków: ?

36. Jeżeli w definicji typu adnotacyjnego umieścimy metaadnotację **@Retention(RetentionPolicy.RUNTIME)** oznaczać to, że będzie ona przetwarzana dopiero w czasie wykonania kodu
Odpowiedź: **Prawda**

38. W Javie **Klasę wewnętrzną** możemy zdefiniować dla:
Klasy konkretnej: **Tak**
Interfejsu: **Nie**
Klasy abstrakcyjnej: **Tak**

39. Napisz w Javie kod serwera dla aplikacji Klient Serwer działającej na Socketach.

Serwer powinien przyłączyć klienta, wysłać klientowi pewien String, rozłączyć go i zakończyć działanie.
Wykorzystaj słowa kluczowe:
`accept, close, flush, getOutputStream, IOexception, java.io, java.net, println, PrintWriter, ServerSocket, Socket`

Odpowiedź: **//kod ze zdj, ocenione na 3 z 3**

```
try{
	// utworzenie gniazda
	String serverHost = …;
	int serverPort = …;

	Socket socket = new Socket(serverHost, serverPort)

	OutputStream sockOut = socket.getOutputStream();
	InputStream sockIn = socket.getInputStream();

	socktOut.write(...);
	socktIn.read();

	sockOut.close();
	sockIn.close();
	socket.close();
} catch(UnknownHostException exc) {
	// nieznany host
} catch(SocketException exc) {
	// wyjątki związane z komunikacją przez gniazda
} catch(IOException exc) {
	// inne wyjątki we/wy
}
```

40. Napisz funkcję przyjmującą tablicę stringów (zakładamy że ten argument != null i ma długość > 0) i zwracającą string który występuje w tej tablicy co najmniej dwa razy. Jeśli takiego nie ma zwróć null

```
public String foo(String args[])
{
	HashSet<String> set = new HashSet<>();
	for(String s : args)
		if(set.contains(s))
			return s;
		else
			set.add(s);
	return null;
}
```

41. Słowo kluczowe “this” może być użyte w ciele konstruktora.

Odpowiedź: **Prawda**

42. Jaki napis wypisze poniższy fragment kodu?

```
Integer i = 0;
System.out.println(i != null  ?  1<<2  : 6 >> 1);
```

Odpowiedź: **4**
Objaśnienie:
0b100=2^2 = 4
0b1000=2^3 = 8
0b10000=2^4 = 16

44. Słowa kluczowe używane przy wyjątkach to:
try, catch, ***, throw, ***
Odpowiedź: **throws, finally**

46. Można stworzyć konstruktor klasy abstrakcyjnej.
Odpowiedź: **Prawda**

47. Obiekt w Javie może dziedziczyć stan z kilku KLAS bazowych.
Odpowiedź: **Fałsz**

48. Przyporządkuj nazwy strumieni wejściowych do odpowiednich kategorii.
Każdy użyj tylko raz, pewne mogą pozostać niewykorzystane.

Konwersja char - byte: **OutputStreamWriter -- do sprawdzenia!!**
Strumień obiektów, np Date: **ObjectInputStream**
Odczyt linii pliku tekstowego: **BufferedReader**

49. Metodę domyślną interfejsu w javie oznaczamy słowem
Odpowiedź: **default**

Dodatkowe info Java

Jeżeli w definicji typu adnotacyjnego umieścimy metadnotację **@Retention(RetentionPolicy.RUNTIME)** oznacza to że będzie ona przetwarzana dopiero w momencie wykonywania kodu

W języku java plik źródłowy musi mieć tę samą nazwę co zawarta w nim klasa publiczna, natomiast klasa niepubliczna może mieć inną.

W javie 8 adnotacją możemy opatrzeć argument metody

W javie klazulę extends możemy umieścić w nagłówku dla: klasy konkretnej, klasy abstrakcyjnej, interfejsu.

Strumienie wejsciowe:
Binarny odczyt liczby float: **DataInputStream**
Strumień bajtów: **FileInputStream**
Strumien znaków: **BufferedReader**

W javie klasę wewnętrzną można zdefiniować dla: klasy konkretnej, klasy abstrakcyjnej, interfejsu

`@Override` - adnotacja metody, wykorzystywana podczas dziedziczenia, na przykład klasa Object ma metodę hashCode(), i w naszym obiekcie można ją nadpisać.
`@Deprecated` - porzucona funkcjonalność, może się popsuć w kolejnej wersji
`@Test` - przy testach JUnit
`@SuprressWarnings` - ignoruje ostrzeżenia

Tablice nie mogą mieć generycznych typów:
```
Pair<String>[] table = new Pair<>[10]; // !!!
```

`Collection<? extends Andrzej>` - kolekcja zawierająca typy implementujące interfejs Andrzej / dziedziczące z klasy Andrzej

Klasy strumieni wejścia/wejścia
- bazowe klasy abstrakcyjne: `InputStream, OutputStream`
- sekwencje bajtów: `FileInputStream, FileOutputStream`
- sekwencje dowolnych typów danych: `DataInputStream, DataOutputStream`
- obiekty: `ObjectInputStream, ObjectOutputStream`

Klasy operacji na tekście: `FileReader, FileWriter`
Obsługa niestandardowego kodowania: opakowanie FileReader w InputStreamReader
Odczyt z konsoli: `Scanner`

Serializacja: implementacja interfejsu `Serializable`, metody readObject() i writeObject() w obiekcie strumienia

Programowanie sieciowe

Adresowanie localhosta: InetAddress a0 = InetAddress.getLocalHost();
Host odległy: InetAddress address = InetAddress.getByName("www.oreilly.com");

Nie uwierzytelniony aplet nie może wykonywać operacji **getByName** i **getAllByName**!

Strumienie filtrujące oraz transformujące są przykładem **dekoratorów**

PrintWriter jest właściwą klasą do zapisu plików tekstowych!
Klasy **InputStreamReader** oraz **OutputStreamWriter** pozwalają na przezroczystą dla programisty konwersję źródłowego strumienia bajtowego na znaki Unicode i na odwrót.

Gniazda umożliwiają niskopoziomową komunikację sieciową. Gniazdo to jeden z końców dwukierunkowego połączenia pomiędzy klientem a serwerem.

Pakiet datagramowy (zwykle UDP) jest reprezentowany przez klasę **DatagramPacket.**

Gniazda bezpołączeniowe (ang. datagram sockets) umożliwiają przesyłanie i odbieranie datagramów. Najczęściej są to pakiety UDP. Przy tym połączeniu nie rozróżniamy klienta i serwera, ponieważ połączenie nie jest zestawiane

(RPC) Remote Procedure Call - Zdalne wywołanie procedur
(RMI) Remote Method Invocation - Zdalne wywoływanie metod
System RMI pozwala na to, aby obiekt działający na jednej maszynie wirtualnej wywoływał metody obiektu działającego na innej maszynie wirtualnej Javy

Protokół RMI zapewnia,że obiekty zdalne mogą być zniszczone dopiero, kiedy nie ma do nich ani zdalnych, ani lokalnych referencji.
RMI do przekazywania obiektów przez sieć wykorzystuje mechanizm serializacji.

Głównym problemem przy RMI i zarządzaniu pamięcią jest rozproszone zwalnianie pamięci **(distributed garbage collection)**. Rozwiązaniem tego jest licznik referencji dla każdego obiektu zdalnego.

RMI pozwala na szybkie i łatwe tworzenie aplikacji rozproszonych.
Ze względu na silny związek z Javą RMI jest w stanie współdziałać z dziedziczeniem.
Wywołania RMI są blokujące, jak zwykle w RPC, jednak wbudowana wielowątkowość pozwala na uruchomienie innych funkcji programu w trakcie obliczeń na maszynie zdalnej

Bazy danych

Rodzaje sterowników JDBC (Java Database Connectivity)

1) mostek JDBC-ODBC
2) mieszany kod rodzimy klienta bazy danych i Javy
3) czysty kod Javy komunikujący się niezależnym od bazy danych protokołem sieciowym, tłumaczonym przez pośredni serwer
4) czysty kod Javy komunikujący się zależnym od bazy danych protokołem sieciowym

Metody obiektu typu ResultSet do pobierania wartości pobranych rekordów

`first()` – przejście do pierwszego rekordu
`last()` – przejście do ostatniego rekordu
`next()` – przejście do następnego rekordu
`previous()` – przejście do poprzedniego rekordu
`getFloat("nazwa kolumny")` – pobranie wartości float
`getInt("nazwa kolumny")` – pobranie wartości integer
`getDate("nazwa kolumny")` – pobranie wartości typu Date
`getString("nazwa kolumny")` – pobranie wartości String

Zadania Scala

1. Co zwróci następująca konstrukcja (proszę zaznaczyć też typ):
```
for( i <- 0 untril 4 by 2; c <- “ab”) yield (c+i).toChar
```

Napisz w scali wyrażenie tworzące zmienną x o wartości -1 jeśli y > 0, “!!!” w przeciwnym wypadku. Jaki będzie typ x?
```
def manOf[T: Manifest](t: T): Manifest[T] = manifest[T]
var y = -20
var x = if (y > 0) -1 else "!!!"
println(manOf(x))
```
Typ: Any

2. W języku Scala
   1. Wartość(value) b jest ArrayBufferem.
      Co oznacza kod: b.sortWith(_>_)?
   2. Mamy następujące dwie klasy w Scali:
      class Osoba1 {var wiek = 20} oraz class Osoba2 {val wiek = 20}
      Czy można ustawić wiek obiektu klasy Osoba1 i Osoba , a jeśli tak to w jakis sposób?

Odpowiedź:
1. **Oznacza to, że należy posortować wartości malejąco.**
2. **Nie można ustawić wieku osoby klasy Osoba2 gdyż val jest niemutowalną zmienną.**
   Można natomiast ustawić wiek Osoba1;
   val os: Osoba1 =new Osoba1();
   os.wiek = 30;

3. Co zwróci następująca konstrukcja:
```
for(c<- “abc”; i <- 1 to 2) yield ( c+i).toChar
```
Odpowiedź: bccdde

4. Napisz w Scali kod zapisujący w kolekcji listowej (np. Vector) wszystkie pary liczb

(i,j), 1<= i, j <= 5 których suma jest nie nieparzysta, Użyj for:
Odpowiedź: var e = for(i <- Range(0, 2); j <- Range(0, 5) if (i + j) % 2 == 1) yield (i, j)

5. Jak jest różnica między
`val a = myArray(10)`
a
`val a = new myArray(10)`
Zakładając że myArray to jakiś typ tablicowy

Odpowiedź: **Pierwsza wersja: używa metody apply z obiektu towarzyszącego klasy myArray. Drugie wywołuje konstruktor**

6. Co zwróci następująca konstrukcja (proszę zaznaczyć też typ):
`for(i <- 0 until 4 by 2; c <- “ab”) yield (c + i).toChar`
Nie znam scali, więc nie wytłumaczę dlaczego, ale: Vector(a, b, c, d)

7. Podaj po jednym przykładzie kolekcji mutowalnej i niemutowalnej z biblioteki standardowej scali
Mutowalne:

- MutableList
- Arraybuffer
- Listbuffer
- Stringbuilder
- Linkedlist
- Double linked list
- mutable.Queue
- Nie chce mi się pisać więcej https://www.scala-lang.org/api/2.12.2/scala/collection/mutable/index.html

Niemutowalne:

- List
- Stream
- Vector
- Stack
- Queue
- Range
- Nie chce mi się pisać więcej https://www.scala-lang.org/api/2.12.2/scala/collection/immutable/index.html

w Javie **List<Integer> NIE JEST** podtypem **List<Number>.**

W Kotlinie **JEST**!

Dodatkowe info Scala - same ciekawostki, lub rzeczy które mogą się przydać

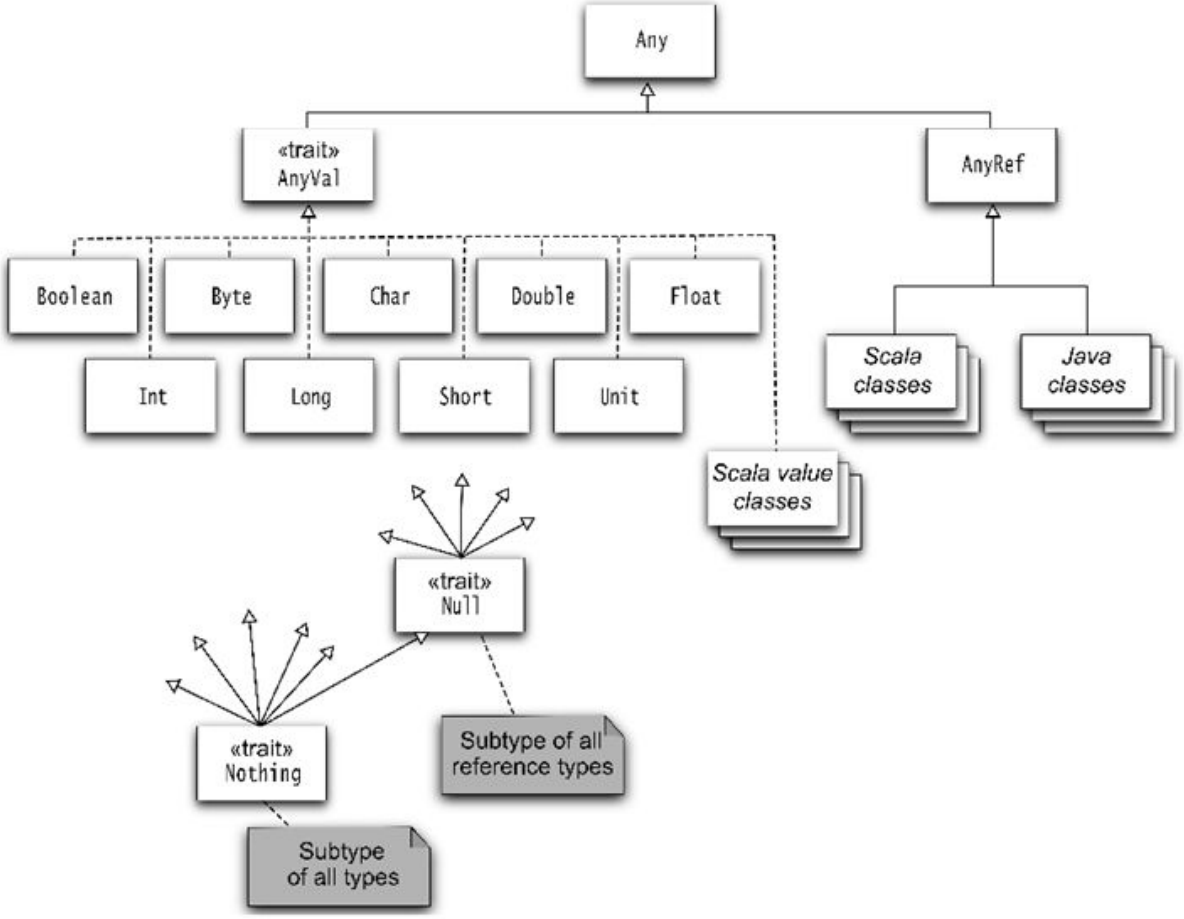
val - value - immutable(niezmienny)
var - variable - muttable(zmienny)
Scala od wersji 5 domyśla się typu.
jednak możliwe jest określenie typu odgórnie np:
val x:Int = 6
val y: Double = -3.2

Vector - takie samo jak w Javie ArrayList ale “niezmienny”

brevity(zwięzłość)

- return może zostać pominięty, wynikiem jest ostatnie wyrażenie złożone -- do sprawdzenia
- średniki mogą zostać pominięte
- “parens”(chyba chodzi o nawiasy ()) w wywołaniach mogą zostać pominięte jeśli metoda ma 0 lub 1 argumentów - tylko w tak zwanej notacji operatora
- tak samo z kropką po nazwie obiektu, a przed metodą - w takim samym przypadku jak wyżej(notacji operatora?)

Metody bez parametrów
dobrym stylem jest używać () dla mutatorów(getów)
i nie używać () dla accesorów(setter)



Typy Scali

Klasy Scala Any, AnyRef i AnyVal nie pojawiają się jako klasy w kodzie bajtowym - z powodu ograniczeń JVM. (Java Virtual Machine)

W Scali wszystko jest obiektem(w javie nie), JVM nie może ogarnąć tego i wygenerowany kod bajtowy nie pokazuje tego.

W Scali wszystkie obiekty są podklasą **Any.**
AnyRef w Scala jest odpowiednikiem java.lang.**Object** (przynajmniej na JVM).
AnyVal w Scali są odpowiednikiem typów prymitywnych w javie

Typ **Nothing** nie dziedziczy z żadnego innego typu

Lists jak i **String** są niezmienne(immutable)
dostęp do początku/konca z O(1) reszta operacji O(n)

Listy są konstruowane od prawej do lewej
Indexing: list(0)
Slicing: list.slice(1, 3); list.slice(2, list.last)
Reversing: list.reverse
Sorting: list.sorted

=> nieformalnie operator rakiety
pętla for:

```
for (y <- List(1, 2, 3))
{
      println(y)
}

for (x<- List(1,2,3); y<-List(4,5))
yield x * y
```

wynik:

```
List(4, 5, 8, 10, 12, 15)
for{  x <- 1 to 7 // generator
      y = x%2;    // definicja?
      if( y == 0) // filtr
   } yield  {
      println(x)
      x
}
```

Wynik: 2 4 6

dla pętli for możliwość używania wielu filtrów
generowana kolekcja jest kompatibilna z pierwszym podanym generatorem

Funkcja operatora(operator function)

```
def ~=(x: Double, y: Double, precision: Double) = {
  if ((x - y).abs < precision) true else false
}
```

wywołanie:
~=(a, b, 0.0001)
wynik: **true**

Tablice są zmienne(mutable)

Sortowanie tablicy(ale nie arrayBuffer)

```
val a = Array(1, 7, 2, 9)
scala.util.Sorting.quickSort(a)
//  Array(1, 2, 7, 9)
// generalnie działa ze wszystkimi typami  z definiowanym comparison op

ArrayBuffer("Mary", "had", "a", "little", "lamb").max
// "little"
```

++ łączy(concatenates) dwie kolekcje

Sealed classes równoważne z klasą finalna
Scala **##** prawie takie samo jak w Javie **hashCode**
Scala **==** odpowiednik w Javie **equals**
aby sprawdzić referencje dwóch obiektów trzeba użyć **eq**

Gdyby pojawiło się pytanie z tego tematu z Kotlina, to tam:
== jest odpowiednikiem funkcji equals() (*equality*). Zanegowane: **!=**
=== porównuje referencje, tak jak w JS (*identity*). Zanegowane: **!==**
I nie ma operatorów **>>, >>>, <<, &, |, ^**. Jest kolejno: **shr, ushr, shl, and, or, xor**
I zmienne deklaruje się dokładnie tak jak w Scali.

Dodatkowe info Kotlin

Standardowe info o tym na co komu kotlin:

- Niby szybko się kompiluje
- Możesz używać **;** ale nie musisz jak java scripcie
- Jest na kompy, serwery i androida
- null-pointer safety
- Brak typów prymitywnych tylko obiekty
- **fun main()** może być bez parametrów
- **a: Int** w ten sposób deklarujemy
- **val n = BigInteger(12345678)** są automatyczne typy (działa **val** i **var**) a nazywa się to Inferencja typów czyli, że sam se wylicza na podstawie tego co przyrównujesz
- **var a = 10L** to jest long
- **val ulong = 42uL** to jest long bez paru znaków
- **toByte() toShort() toInt() toLong() toFloat() toDouble() toChar()**
- **==** oraz **!=** działa po ludzku nie jak w jave nie trzeba używać **equals()**
- mamy za to **===** oraz **!==** do porównywania referencji
- są **<**, **>**, **<=**, **>=**
- są **||** i **&&**
- ale nie ma operatorów logicznych | i & zamiast tego jest **or** i **and**
- są **shr** i **shl** czyli przesunięcia bitowe odpowiednio >> i <<

Hello world w kotlinie

```
fun main(args: Array<String>) {
      println("Hello, World!")
}
```

Kompilacja w kotlinie

```
kotlinc nazwa.kt
```

Odpalanie programu
`kotlin Nazwakt`

Pętle

- for
- while
- do-while
- foreach
- repeat

```
sth.forEach {
      print(it.toString())
}
repeat(10) { i -> println("Jesteśmy w ${i+1}-wszej linii.") }
```

Tablice

```
val arr = arrayOf(1, 2, 5)
val squares = Array(10, { i -> i * i })
```

tablica w kotlinie to nie element języka tylko klasa kolekcji

niemutowalne oznacza read-only

Lambda

```
val allowedUsers = users.filter { it.age > MINIMUM_AGE }
```

to jest lambda
fajna nie
it jest domyślnym iteratorem jak własnego nie zadeklarujesz

zamiast tego możesz tak:

```
val allowedUsers = users.filter { x -> x.age > MINIMUM_AGE }
```

I cyk kolekcja:

```
val persons = listOf(Person("Max", 18), Person("David", 12),
Person("Peter", 23), Person("Pamela", 23))
```

List (interfejs) -- lista niemutowalna (zawiera size, get etc.). Podobnie Set, Map.