1. **Чим відрізняється посилання Java від указівника C++?**

У Java немає типів указівників. Імена змінних непримітивних типів по суті є іменами посилань на відповідні об'єкти.

Значение объекта это ссылка на память где объект лежит, то есть неявно это как бы указатель

1. **Чим типи-посилання відрізняються від типів-значень?**

Типы значений:

* Целочисленные типы (byte, sbyte, char, short, ushort, int, uint, long, ulong)
* Типы с плавающей запятой (float, double)
* Тип decimal
* Тип bool
* Перечисления enum
* Структуры (struct)

Ссылочные типы:

* Тип object
* Тип string
* Классы (class)
* Интерфейсы (interface)
* Делегаты (delegate)

1. **Як у Java здійснюється розіменування?**

Розіменування не потрібне: звертання до примітивних типів завжди здійснюється за значенням, а до непримітивних - по посиланню

1. **Що є результатом присвоєння одного посилання іншому?**

Присвоювання значення одного посилання іншому не забезпечує копіювання об'єктів. Після присвоювання два посилання посилаються на один об'єкт.

1. **Чи завжди змінним типів-посилань можна присвоювати null?**

Константа **null**може бути присвоєна змінній будь-якого типу-посилання

1. **Як у Java видалити об'єкт, який було створено за допомогою new?**

На відміну від С++, звільнення пам'яті від непотрібних об'єктів не потрібно. У Java немає операції **delete**. Для звільнення пам'яті використовується спеціальний механізм, який має назву збирання сміття

1. **У чому полягає "збирання сміття"?**

Цей механізм базується на підрахунку посилань на об'єкти. Кожен об'єкт має свій лічильник посилань. Коли посилання копіюється в нову змінну типу-посилання, лічильник збільшується на одиницю. Коли посилання виходить з області видимості, чи перестає вказувати на об'єкт, лічильник зменшується на одиницю. Коли віртуальній машині Java не вистачає оперативної пам'яті, запускається збирач сміття. Він переглядає список об'єктів і видаляє з пам'яті всі об'єкти, для яких кількість посилань дорівнює 0.

1. **Чи можна використовувати змінні для визначення довжини масиву?**

Розмір масиву не є частиною типу, як це реалізовано в C++. Це дозволяє визначити масив необхідного розміру при виконанні програми

1. **Чи можна змінити розміри масиву за допомогою поля length?**

Це поле доступне тільки для читання:

**int**[] a = **new int**[10];

System.out.println(a.length); // 10

1. **Як додати новий елемент у кінець масиву?**

array[array.length()-1] = 123;

1. **Як визначити кількість стовпців двовимірного масиву?**

array[0].length

1. **Чи можна створити двовимірний масив з різною довжиною рядків?**

нет

1. **Чим відрізняється застосування двох різних конструкцій for для обходу елементів масиву?**

Альтернативна форма циклу **for** (починаючи з JDK 1.5) дозволяє спростити повний обхід масивів. Наприклад, замість

**int**[] nums = {1, 2, 3, 4, 5, 6};

**for** (**int** i = 0; i < nums.length; i++) {

System.out.println(nums[i]);

}

можна написати

**int**[] nums = {1, 2, 3, 4, 5, 6};

**for**(**int** n : nums) {

System.out.println(n);

}

1. **Чим визначається розмір масиву, який створюється функцією arraycopy()?**

a\_from + size

1. **Чи можна за допомогою arraycopy() скопіювати частину масиву?**

System.arraycopy(a, a\_from, b, b\_from, size);

1. **Як здійснити читання масиву з клавіатури?**

System.out.println("Уведіть елементи масиву:");

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

a[i] = s.nextDouble();

}

1. **У які способи можна заповнити елементи масиву без циклу?**

Для заповнення масивів можна використовувати статичні методи класу Arrays, реалізованого в пакеті java.util. Для всіх примітивних типів і типу Object реалізовані функції fill() у двох видах:

**public static void** fill(*тип*[] a, *тип* val)

**public static void** fill(*тип*[] a, **int** fromIndex, **int** toIndex, *тип* val)

Тут *тип* означає один з фундаментальних (примітивних) типів або тип Object. Перший варіант використовується для заповнення всього масиву, другий - частини, при чому елемент з номером toIndex не включається в послідовність

1. **Як без циклу встановити, що елементи масивів збігаються?**

**public static boolean** equals(*тип*[] a, *тип*[] a2)

Тут *тип* означає один з фундаментальних (примітивних) типів або тип Object. Масиви порівнюються поелементно. Два масиви також вважаються еквівалентними, якщо обидва посилання - **null**

Є також метод deepEquals(), використання якого аналогічне. Різниця є істотною для багатовимірних масивів. Здійснюється більш "глибока" перевірка

1. **Чи можна без циклу відсортувати частину масиву?**

За допомогою функції sort()можна здійснити сортування масиву чисел за зростанням

1. **Чи дозволяє функція binarySearch() здійснити пошук у невідсортованому масиві?**

no

1. **Чи можна змінити значення елементів масиву за допомогою функції?**

yes

1. **Як у Java створити функцію зі змінною кількістю аргументів?**

static void printIntegers(int... a) {

for (int i = 0; i < a.length; i++)

System.out.println(a[i]);

}

1. **З яких основних елементів складається опис класу?**

*Клас*-це структурований тип даних, набір елементів даних різних типів і функцій для роботи з цими даними. Опис класу складається зі специфікаторів (наприклад, **public**, **final**), імені, імені базового класу, списку інтерфейсів і тіла у фігурних дужках.

1. **Чи завжди необхідно явно ініціалізувати поля класу?**

Тіло класумістить поля (їм відповідають елементи даних у C++) і методи (функції-елементи в C++). Поля і методи разом іменуються елементами (членами) класу. Нижче наводиться приклад опису класу.

Нет, не всегда.

1. **Чи можна в Java поза класом реалізовувати методи, оголошені всередині класу?**

Методи завжди реалізуються усередині визначення класу.

1. **Чим відрізняються статичні та нестатичні елементи класу?**

Звертання до таких полів і методів може здійснюватися без створення екземпляра класу. Статичні методи не отримують посилання на об'єкт і не можуть використовувати посилання **this**.

Звертання до статичних елементів може здійснюватися як через ім'я класу, так і через посилання на об'єкт:

SomeClass.x = 30;

SomeClass s = **new** SomeClass();

s.x = 40;

1. **Як здійснюється ініціалізація статичних даних?**

Статичні поля можуть бути проініціалізовані при описі:

**class** SomeClass {

**static double** x = 10;

**static int** i = 20;

}

1. **Де може бути розташована конструкція ініціалізації?**

усередині тіла класу. Такий блок буде виконуватися щораз при створенні нового об'єкта:

**class** Rectangle {

**double** width;

**double** height;

{

width = 10;

height = 20;

}

**double** area() {

**return** width \* height;

}

}

1. **Як у Java визначається дружній доступ до елементів класу?**

Доступ до закритих (**private**) елементів класу обмежений методами усередині класу. У Java немає ключового слова **friend**, яке забезпечує доступ до закритих елементів усередині класу.

1. **Як встановити рівень доступу для групи елементів?**

Як і C++, Java підтримує закритий (**private**), захищений (**protected**) і відкритий (**public**) рівні доступу. Сам клас може бути оголошений як **public**. На відміну від C++, Java вимагає окремої специфікації доступу для кожного елемента, або групи полів одного типу:

**public class** SomeClass {

**private int** i;

**private double** x, y; // Два поля одного типу

**public void** setI(**int** value) {

i = value;

}

**public int** getI() {

**return** i;

}

**public void** setX(**double** value) {

x = value;

}

**public double** getX() {

**return** x;

}

}

1. **У чому полягає зміст інкапсуляції та як вона реалізована в Java?**

*Інкапсуляція*(приховування даних) - одна з трьох парадигм об'єктно-орієнтованого програмування. Зміст інкапсуляції полягає у приховуванні від зовнішнього користувача деталей реалізації об'єкту. Зокрема доступ до даних (полів), які зазвичай описані з модифікатором **private**, здійснюється через відкриті функції доступу. Як правило, це так звані сеттери та геттери. Якщо поле має ім'я name, відповідні функції доступу мають імена setNameта getName.

1. **Як можна використовувати посилання this?**

Ключове слово **this**використовувати явно, наприклад, коли треба повернути з функції посилання на поточний об'єкт, або запобігти конфлікту імен**.**

1. **Як викликати конструктори з інших конструкторів?**

Один конструктор можна викликати з іншого з використанням слова **this**, після якого випливають необхідні аргументи.

**public class** Rectangle {

**private double** width;

**private double** height;

**public** Rectangle(**double** width, **double** height) {

**this.**width = width;

**this.**height = height;

}

**public** Rectangle() {

**this**(10, 20); // виклик іншого конструктора

}

}

1. **Скільки конструкторів без параметрів може бути створено в одному класі?**

У класі може бути визначено кілька конструкторів. Якщо жоден конструктор явно не визначений, автоматично створюється конструктор за умовчанням (без параметрів). Такий конструктор ініціалізує всі поля початковими значеннями за умовчанням.

1. **Як створити клас, у якому немає жодного конструктора?**

Якщо жоден конструктор явно не визначений, автоматично створюється конструктор за умовчанням (без параметрів)

1. **Чому в Java немає деструкторів?**

Можна створити спеціальний метод finalize(), який викликається збирачем сміття перед ліквідацією об'єкта. У деяких випадках об'єкт може бути не вилучений збирачем сміття ніколи (пам'яті вистачало до кінця програми), отже метод finalize()може бути ніколи не викликаний.

Удаление объекта происходит неявно, когда сборщик мусора обнаружит, что на него не осталось ссылок. Тогда он вызовет у него finalize(), но когда именно это произойдет и произойдет ли вообще, система не гарантирует

1. **Коли викликається метод finalize()?**
2. **Чи можна змінити вміст раніше створеного рядка?**

Для модифікації вмісту рядків використовуються стандартні класи StringBufferі StringBuilder. Для поділу рядку на лексеми використовують клас StringTokenizer.

1. **Як змінити конкретний символ у рядку?**

StringBuilder myName = new StringBuilder("domanokz");

myName.setCharAt(4, 'x');

1. **У чому є недоліки й переваги класу StringBuffer у порівнянні з класом String?**

Використання StringBufferможе підвищити ефективність роботи програми у випадках, коли певний рядок зазнає багаторазових модифікацій протягом роботи програми. Але важливо пам'ятати, що декілька посилань вказують на один об'єкт типу StringBuffer. Тому, коли ми його змінюємо, усі посилання вказуватимуть на змінений рядок.

1. **Як перевести число в його рядкове представлення і навпаки?**

**byte[] bytes = { 49, 50, 51, 52 };**

String s2 = **new** String(bytes);

System.out.println(s2); // 1234

**double** d = 1.1;

String sd = String.valueOf(d); // "1.1"

static int parseInt(String s)

static int parseInt(String s, int radix)

1. **Які є недоліки й переваги застосування об'єктів класу Integer замість змінних типу int?**

Класи-обгортки Integer, Double, Boolean, Character, Float, Byte, Shortі Longвикористовують для зберігання даних примітивних типів у об'єктах, до яких можна звертатися посилання. Крім того, ці класи надають ряд корисних методів для перетворення даних.

Недоліком використання типів-обгорток є зменшення ефективності за рахунок додавання операцій розміщення у динамічній пам'яті. Але перевагою є можливість використання значення **null**. Наприклад, якщо значення функції не може бути обчислене, функція може повернути **null**: