1. У яких випадках доцільно використовувати композицію класів?

Композиція класів у цьому випадку припускає створення об'єктів безпосередньо (у тілі класу при оголошенні полів) чи в конструкторах

1. Чи можна в Java цілком розмістити один об'єкт усередині іншого об'єкта?

Ні

1. У чому полягає зміст успадкування?

*Механізм успадкування* полягає в породженні похідних класів від базових. Якщо один клас (похідний) є нащадком іншого (базового), то спадкоємець має можливість безпосередньо користуватися даними і функціями, визначеними в базовому класі. Відносини між класами і підкласами (нащадками) називаються *ієрархією спадкування*класів.

1. У чому є сенс наявності спільного базового класу?

Використання спільних методів та елементів.

1. Які елементи базового класу не успадковуються?

Позначені модифікатором private

1. Як здійснити ініціалізацію базового класу?

Как обычно, от обычной инициализации отличается только инициализация производных классов.

1. Як викликати однойменний метод базового класу з похідного?

super.method();

1. **Де і для чого можна застосовувати ключове слово super?**

Ключове слово **super** використовується для доступу до елементів базового класу з похідного класу, зокрема:

* для виклику перекритого методу базового класу;
* для передачі параметрів конструктору базового класу.

1. **Як перекрити метод з модифікатором final?**

ніяк

1. Чи допускається множинне успадкування класів?

На відміну від C++, у Java дозволяється тільки одиничне успадкування - у класу може бути тільки один базовий клас.

1. Чи можна неявно приводити посилання на базовий клас до посилання на похідний клас?

Зворотне приведення необхідно робити явно:

Base b = **new** Base();

Derived d = (Derived) b;

1. Які можливості надає використання поліморфізму?
2. Чим віртуальна функція відрізняється від невіртуальної?

У мовах об'єктно-орієнтованого програмування пізнє зв'язування реалізується через механізм віртуальних функцій. *Віртуальна функція* (віртуальний метод, virtual method) - це функція, визначена в базовому класі, та перекрита у похідних, так, що конкретна реалізація функції для виклику буде визначатися під час виконання програми.

1. Як у Java указати, що функція віртуальна?

 У Java усі функції є віртуальними, за винятком конструкторів, статичних (**static**), фінальних (**final**) і закритих (**private**) методів. На відміну від C++, слово virtual не використовується.

1. **Чи можна в класах, описаних як final, створювати віртуальні функції?**

Все функции по умолчанию виртуальный, но класс, описанный как final не наследуется, поэтому затрудняюсь ответить

1. **Чому функції з модифікатором private не є віртуальними?**

Их нельзя перекрыть

1. Чи можна створити абстрактний клас без абстрактних методів?

так

1. Чи можуть абстрактні класи містити неабстрактні методи?

Від абстрактного класу не вимагають обов'язкової наявності абстрактних методів

1. У чому перевага інтерфейсів у порівнянні з абстрактними класами?

Клас може реалізувати кілька інтерфейсів.

1. Чи можуть інтерфейси містити поля?

Інтерфейси можуть містити поля, що є фінальними і статичними (константами часу компіляції).

1. **Як створити інтерфейс із закритими (private) методами?**

Методи, визначені в інтерфейсі, є за умовчанням абстрактними і відкритими. У класі, що реалізує інтерфейс, такі методи повинні бути оголошені як **public**.

1. Чи допускається множинне успадкування інтерфейсів?

Інтерфейс може мати кілька базових інтерфейсів:

**interface** Int1 {

**void** f();

**int** g(**int** x);

}

**interface** Int2 {

**void** h(**int** z);

}

**interface** Int3 **extends** Int1, Int2 {

}

Клас може реалізувати кілька інтерфейсів:

**interface** Int1 {

**void** f();

**int** g(**int** x);

}

**interface** Int2 {

**void** h(**int** z);

}

**class** SomeClass **implements** Int1, Int2 {

**public** **void** f() {

}

**public** **int** g(**int** x) {

**return** x;

}

**public** **void** h(**int** z) {

}

}

1. Яким вимогам повинен відповідати клас, який реалізує інтерфейс?

 ім'я інтерфейсу повинне бути поміщене в список реалізованих інтерфейсів. Такий список випливає після ключового слова **implements**. Методи, визначені в інтерфейсі, є за умовчанням абстрактними і відкритими. У класі, що реалізує інтерфейс, такі методи повинні бути оголошені як **public**:

1. Чи може клас реалізовувати кілька інтерфейсів?

Клас може реалізувати кілька інтерфейсів:

**interface** Int1 {

**void** f();

**int** g(**int** x);

}

**interface** Int2 {

**void** h(**int** z);

}

**class** SomeClass **implements** Int1, Int2 {

**public** **void** f() {

}

**public** **int** g(**int** x) {

**return** x;

}

**public** **void** h(**int** z) {

}

}

1. У чому полягає процес клонування об'єктів?

Якщо нам необхідно поелементно скопіювати деякий об'єкт, необхідно використовувати механізм так званого клонування.

У базовому класі java.lang.Object є функція clone(), усталене використання якої дозволяє скопіювати об'єкт поелементно. Ця функція також визначена для масивів, рядків і інших стандартних класів. Наприклад, так можна отримати копію існуючого масиву і працювати з цією копією:

1. **Для чого використовується перевизначення функції equals()?**

Для класів, полями яких є типи-значення, метод класу Objectзабезпечує поелементне порівняння. Якщо ж полями є посилання на об'єкти, необхідно явно перевизначити функцію equals()

1. Як здійснити порівняння елементів масивів?

Для порівняння двох масивів доцільно викликати статичну функцію equals() класу Arrays. Ця функція порівнює елементи масивів (викликає метод equals()):

Arrays.equals(array1, array2);

1. Яким вимогам повинен задовольняти об'єкт, щоб масив таких об'єктів можна було сортувати без визначення ознаки сортування?

Статична функція sort() класу java.util.Array реалізована для масивів усіх примітивних типів. Аналогічно можна реалізувати сортування об'єктів класів, для яких визначене натуральне порівняння, тобто реалізований інтерфейс Comparable.

1. Як визначити спеціальне правило для сортування елементів масиву?

Статична функція sort() класу java.util.Array реалізована для масивів усіх примітивних типів. Аналогічно можна реалізувати сортування об'єктів класів, для яких визначене натуральне порівняння, тобто реалізований інтерфейс Comparable. Єдиний метод цього інтерфейсу - compareTo():

**public int** compareTo(Object o)

Якщо ми не хочемо (чи не можемо) визначити функцію compareTo(), можна створити клас, що реалізує інтерфейс Comparator. Посилання на об'єкт такого класу передаються в як другий (четвертий) параметр функції sort():

**public static void** sort(Object[] a, Comparator c) // Опис функції у Java 2

**public static void** sort(Object[] a, **int** fromIndex, **int** toIndex, Comparator c) // Опис функції у Java 2

Інтерфейс Comparator містить опис методу compare() з двома параметрами. Функція повинна повернути від'ємне число, якщо перший об'єкт під час сортування необхідно вважати меншим, чим інший, значення 0, якщо об'єкти еквівалентні, і додатне число в протилежному випадку.

1. У чому є сенс застосування анотацій?

Анотації дозволяють включити в програмний код додаткову інформацію, яка не може бути визначена за допомогою засобів мови. У тексті програми анотації починаються із символу @. Типовий приклад анотації - @Override. Завдяки цій анотації компілятор може перевірити, чи дійсно відповідний метод був оголошений у базових класах.