1. **Як звернутися до локального класу ззовні блоку?**

До внутрішніх класів також відносяться локальні. До таких класів не можна звернутися ззовні блоку, у якому вони визначені.

1. **Чи може в одному файлі з вихідним текстом бути визначене більш одного відкритого класу?**
2. **Чи можна створювати об'єкт нестатичного внутрішнього класу, не створюючи об'єкту обхопного класу?**

для створення об'єкта внутрішнього класу обов'язково мати в наявності об'єкт обхопного класу.

1. **Чи можуть нестатичні внутрішні класи містити статичні елементи?**

Нестатичні внутрішні класи не можуть містити статичних елементів.

1. **Чим відрізняються статичні вкладені класи від внутрішніх?**

Статический могут содержать статические свойства и методы

1. **Чи можуть статичні вкладені класи містити нестатичні елементи?**

Могут содержать любые элементы

1. **Чи можна створювати класи усередині інтерфейсів?**

Вложенные в интерфейсы классы. Вложенный (nested) в интерфейс класс является открытым (public) и статическим (static) даже без явного указания этих модификаторов. Помещая класс внутрь интерфейса, мы показываем, что он является неотъемлемой частью API этого интерфейса и более нигде не используется.

1. **Чи є локальний клас статичним?**

no

1. **Чи є безіменний клас статичним?**

no

1. **Чому не можна створити явний конструктор безіменного класу?**
2. **Що таке лямбда-вираз?**

выражение с переменными параметрами

У мовах програмування є поняття *функціонального об'єкта* - об'єкта, який можна використовувати як функцію. *Лямбда-вираз* - це спеціальний синтаксис опису функціонального об'єкта всередині методу. Іншими словами, лямбда-вираз - це спосіб опису функції всередині іншої функції.

Почему такое название? Много лет назад, когда еще не было никаких компьютеров, логик Алонзо Чёрч хотел формализовать, что значит для математической функции быть эффективно вычисляемой. (Любопытно, что есть функции, которые, как известно, существуют, но никто не знает, как вычислить их значения.) Он использовал греческую букву лямбда (λ), чтобы отметить параметры. Если бы он знал о Java API, он написал бы что-то не сильно похожее на то, что вы видели, скорее всего.  
  
Почему буква λ? Разве Чёрч использовал все буквы алфавита? На самом деле, почтенный труд Principia Mathematica использует символ ˆ для обозначения свободных переменных, которые вдохновили Чёрча использовать заглавную лямбда (Λ) для параметров. Но, в конце концов, он переключился на строчной вариант буквы. С тех пор, выражение с переменными параметрами было названо «лямбда-выражение».

1. **Що таке функціональний інтерфейс?**

Дуже часто інтерфейси в Java містять оголошення однієї абстрактної функції (без усталеної реалізації). Такі інтерфейси отримали назву *функціональних інтерфейсів.* Вони повсюдно використовуються для реалізації механізмів зворотного виклику, обробки подій і т. д. Не дивлячись на їх простоту, для їх реалізації, тим не менш, потрібен окремий клас - звичайний, вкладений або безіменний. Навіть використовуючи безіменний клас ми отримуємо громіздкий синтаксис, який погано читається. Скоротити необхідність безіменних класів у вихідному коді дозволяють лямбда-вирази, які з'явилися у версії Java 8.

Лямбда-выражения имеют обратную совместимость с этими интерфейсами.

Вы можете поставить лямбда-выражение всякий раз, когда ожидается объект интерфейса с одним абстрактным методом. Такой интерфейс называется функциональным интерфейсом.

1. **Які переваги надають лямбда-вирази?**

Лямбда-выражение представляет собой блок кода, который можно передать в другое место, поэтому он может быть выполнен позже, один или несколько раз

Вы никогда не указываете тип результата лямбда-выражения. Это всегда выясняется из контекста. Например, выражение

(String firstStr, String secondStr) -> Integer.compare(firstStr.length(), secondStr.length())

может быть использовано в контексте, где ожидается результат типа int.

1. **Для чого використовуються посилання на методи?**

Дуже часто все тіло лямбда-виразу складається лише з виклику існуючого методу. У цьому випадку замість лямбда-виразу можна використовувати посилання на цей метод. Існує кілька варіантів опису посилань на методи.

Компактность кода

1. **Як створити посилання на нестатичний метод?**

для метода экземпляра требуется соответствующий экземпляр для вызова. Изменить метод на статический или предоставить экземпляр класса

|  |  |  |
| --- | --- | --- |
| Посилання на нестатичний метод для заданого об'єкта | ім'яОб'єкта::ім'яНестатичногоМетоду | s::toString |

1. **Як створити посилання на статичний метод?**

|  |  |  |
| --- | --- | --- |
| Посилання на статичний метод | ім'яКласу::ім'яСтатичногоМетоду | String::valueOf |

1. **Як створити посилання на конструктор?**

|  |  |  |
| --- | --- | --- |
| Посилання на конструктор | **ім'яКласу::new** | **String::new** |

1. **У яких випадках доцільно створювати узагальнені класи?**

Часто виникає необхідність у створенні так званих класів-контейнерів - таких, які містять об'єкти довільних типів. При цьому над елементами контейнерів необхідно виконувати деякі однотипні дії. Код для обробки об'єктів різних типів виглядає практично однаково. Це особливо справедливо, якщо для різних типів даних потрібно реалізувати алгоритми на кшталт швидкого сортування або способи обробки таких структур даних, як зв'язаний список або бінарне дерево. У таких випадках код однаковий для всіх типів об'єктів.  
  
Парадигма узагальненого програмування передбачає опис правил зберігання даних і алгоритмів у загальному вигляді незалежно від конкретних типів даних. Конкретні типи даних, над якими виконуються дії, специфікуються пізніше. Механізми розділення структур даних і алгоритмів, а також формування абстрактних описів вимог до даних, визначаються по-різному в різних мовах програмування. Спочатку можливості узагальненого програмування були представлені в сімдесяті роки XX століття мовами CLU і Ада (узагальнені функції), пізніше були реалізовані в мові ML (параметричний поліморфізм).

1. **Що таке параметризований тип?**

Тип даних з параметром у кутових дужках (наприклад, Pair<String>) має назву параметризованого типу.

1. **Чи можна створювати об'єкти узагальнених типів?**

Під час створення об'єкта узагальненого типу у кутових дужках вказують імена реальних типів. Можна використовувати тільки типи-посилання. Попередній приклад можна реалізувати з застосуванням узагальнень.

1. **Чому в контейнерному класі не можна зберігати цілі і дійсні числа безпосередньо, а можна тільки посилання?**

Поля обобщений работают с ссылками на объекты

1. **Як з масиву отримати список?**

Arrays.asList(1, 2, 3, 4, 5)

1. **Як зберегти у списку цілі і дійсні значення?**

Использовать классы обертки?

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class test {

public static void main(String[] args) {

Integer i = new Integer(4);

Double d = new Double(4.1231);

List<Object> l = new ArrayList<>(Arrays.asList(i,d));

System.out.println(l); // [4, 4.1231]

}

}

1. **У чому перевага опису посилання на інтерфейс контейнера (наприклад, List) у порівнянні з описом посилання на клас, що реалізує контейнер (наприклад, ArrayList)?**

Другий варіант іноді не є бажаним, оскільки в такому випадку знижується гнучкість програми. Перший варіант дозволить легко замінити реалізацію списку ArrayList на будь-яку іншу реалізацію інтерфейсу List, яка більше відповідає вимогам конкретної задачі. У другому випадку є спокуса викликати методи, специфічні для ArrayList, тому перехід на іншу реалізацію буде ускладнено.

1. **У чому перевага використання ітераторів у порівнянні з індексами елементів контейнера?**

Спеціальний вид ітератора списку, ListIterator, надає додаткові можливості ітерації, зокрема, проходження по списку у зворотному порядку. В наступному прикладі для перевірки, чи є слово паліндромом використовується список символів і ListIterator, який забезпечує проходження в зворотному порядку:

* Индексация не подходит для некоторых структур данных, в частности, для структур данных с медленным произвольным доступом или вообще без поддержки такового (например, список или дерево).
* Итераторы предоставляют возможность последовательного перебора любых структур данных, поэтому делают код более читаемым, удобным для повторного использования и менее чувствительным к изменениям структур данных.
* Итераторы могут предоставлять дополнительные возможности при навигации по элементам. Например, проверку отсутствия пропусков элементов или защиту от повторного перебора одного и того же элемента.
* Некоторые контейнеры могут предоставлять возможность модифицировать свои объекты без влияния на сам итератор. Например, после того, как итератор уже «прошёл» первый элемент, можно вставить дополнительные элементы в начало контейнера без каких-либо нежелательных последствий. При использовании индексации это проблематично из-за смены номеров индексов.

1. **Коли доцільніше використовувати ArrayList у порівнянні з LinkedList?**

Когда чаще надо извлекать элементы, чем добавлять. А в целом всегда

1. **Коли доцільніше використовувати LinkedList у порівнянні з ArrayList?**
2. **Чим множина відрізняється від списку?**

Обе структуры используют интерфейс Collection. Множина - це колекція, що не містить однакових елементів. Три основних реалізації інтерфейсу Set - HashSet, LinkedHashSet і TreeSet. Як і списки, множини є узагальненими типами. Класи HashSet і LinkedHashSet використовують хеш-коди для ідентифікації елемента. Клас TreeSet використовує двійкове дерево для збереження елементів і гарантує їх певний порядок.

1. **Наведіть приклади стандартних функціональних інтерфейсів.**

|  |  |
| --- | --- |
| BiConsumer<T,U> | Представляє операцію, яка приймає два вхідних аргументи та не повертає результату |
| BiFunction<T,U,R> | Представляє функцію, яка приймає два аргументи і повертає результат |
| BinaryOperator<T> | Представляє операцію над двома операндами одного типу, виробляючи результат того ж типу, що й операнди |
| BiPredicate<T,U> | Представляє предикат (функцію з результатом типу **boolean**) з двома аргументами |
| BooleanSupplier | Представляє "постачальника" результату типу **boolean** |
| Consumer<T> | Представляє операцію, яка приймає один аргумент і не повертає результату |
| DoubleBinaryOperator | Представляє операцію над двома аргументами типу **double**, яка повертає результат типу **double** |
| DoubleConsumer | Представляє операцію, яка приймає один аргумент типу **double** і не повертає результату |
| DoubleFunction<R> | Представляє операцію, яка приймає один аргумент типу **double** і повертає результат |
| DoublePredicate | Представляє предикат (функцію з результатом типу **boolean**) з одним аргументом типу **double** |
| DoubleSupplier | Представляє "постачальника" результату типу **double** |
| DoubleToIntFunction | Представляє операцію, яка приймає один аргумент типу **double** і повертає результат типу **int** |
| DoubleToLongFunction | Представляє операцію, яка приймає один аргумент типу **double** і повертає результат типу **long** |
| DoubleUnaryOperator | Представляє операцію, яка приймає один аргумент типу **double** і повертає результат типу **double** |
| Function<T,R> | Представляє операцію, яка приймає один аргумент і повертає результат |
| IntBinaryOperator | Представляє операцію над двома аргументами типу **int**, яка повертає результат типу **int** |
| IntConsumer | Представляє операцію, яка приймає один аргумент типу int і не повертає результату |
| IntFunction<R> | Представляє операцію, яка приймає один аргумент типу int і повертає результат |
| IntPredicate | Представляє предикат (функцію з результатом типу **boolean**) з одним аргументом типу **int** |
| IntSupplier | Представляє "постачальника" результату типу **int** |
| IntToDoubleFunction | Представляє операцію, яка приймає один аргумент типу **int** і повертає результат типу **double** |
| IntToLongFunction | Представляє операцію, яка приймає один аргумент типу **int** і повертає результат типу **long** |
| IntUnaryOperator | Представляє операцію, яка приймає один аргумент типу **int** і повертає результат типу **int** |
| LongBinaryOperator | Представляє операцію над двома аргументами типу **long**, що повертає результат типу **long** |
| LongConsumer | Представляє операцію, яка приймає один аргумент типу **long** і не повертає результату |
| LongFunction<R> | Представляє операцію, яка приймає один аргумент типу **long** і повертає результат |
| LongPredicate | Представляє предикат (функцію з результатом типу **boolean**) з одним аргументом типу **long** |
| LongSupplier | Представляє "постачальника" результату типу **long** |
| LongToDoubleFunction | Представляє операцію, яка приймає один аргумент типу **long** і повертає результат типу **double** |
| LongToIntFunction | Представляє операцію, яка приймає один аргумент типу **long** і повертає результат типу **int** |
| LongUnaryOperator | Представляє операцію, яка приймає один аргумент типу **long** і повертає результат типу **long** |
| ObjDoubleConsumer<T> | Представляє функцію, яка приймає аргументи типів T і **double** і не повертає результату |
| ObjIntConsumer<T> | Представляє функцію, яка приймає аргументи типів T і **int** і не повертає результату |
| ObjLongConsumer<T> | Представляє функцію, яка приймає аргументи типів T і **long** і не повертає результату |
| Predicate<T> | Представляє предикат (функцію з результатом типу **boolean**) з одним аргументом |
| Supplier<T> | Представляє "постачальника" результату |
| ToDoubleBiFunction<T,U> | Представляє функцію, яка приймає два аргументи і продукує результат типу **double** |
| ToDoubleFunction<T> | Представляє функцію, яка продукує результат типу **double** |
| ToIntBiFunction<T,U> | Представляє функцію, яка приймає два аргументи і продукує результат типу **int** |
| ToIntFunction<T> | Представляє функцію, яка продукує результат типу **int** |
| ToLongBiFunction<T,U> | Представляє функцію, яка приймає два аргументи і продукує результат типу **long** |
| ToLongFunction<T> | Представляє функцію, яка продукує результат типу **long** |
| UnaryOperator<T> | Представляє операцію над одним операндом, яка повертає результат того ж типу, що й операнд |

Крім перелічених, функціональними інтерфейсами також є узагальнений інтерфейс Comparator, інтерфейс Runnable, який використовують у багатопотоковому програмуванні, а також багато інших.

1. **Як здійснюється композиція лямбда-виразів?**
2. **У чому переваги використання стандартних алгоритмів класу Collections?**

Надежность, простота и т.д.

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

Перегружен метод compareTo интерфейса Comparable или метод Compare интерфейса Comparator для нестандартных классов.