1. **У чому суть RTTI?**

Механізм ідентифікації типу часу виконання (RTTI – run-time type identification) дозволяє визначити тип об'єкта під час виконання програми. Механізм RTTI підтримує більшість мов об'єктно-орієнтованого програмування. Цей механізм, як зазвичай, застосовується до так званих поліморфних типів (класів або структур, які надають віртуальні функції). RTTI дозволяє під час виконання програми визначити точний тип об'єкта за посиланням (або вказівником для мов, що підтримує вказівники). Завдяки розширенню правил сумісності типів посилання на базовий тип може фактично вказувати на об'єкти похідних типів, що є основою поліморфізму. У деяких випадках необхідно явно отримати реальний тип об'єкта, або перевірити, чи ми маємо справу з об'єктом типу, що нас цікавить.

1. **Для чого використовується ключове слово instanceof?**

Определить принадлежность объекта классу

1. **Дайте визначення рефлексії.**

Розглянутий раніше механізм RTTI базується на інформації, яка може бути отримана під час компіляції. Поряд з RTTI, Java надає більш складний і потужний механізм – рефлексію. Цей механізм використовує інформацію, яка може бути недоступна під час компіляції.  
  
Відображення або рефлексія (reflection) – це механізм, який дозволяє програмі відстежувати й модифікувати власну структуру та поведінку під час виконання.

1. **Що за інформація зберігається в полі class?**

Об'єкт типу Class може бути отриманий для існуючих типів (в тому числі і примітивних) з використанням ключового слова class:  
  
Class<?> c1 = String.class;  
Class<?> c2 = double.class;

1. **Як здійснюється завантаження класів в Java?**

Однією з основних особливостей платформи Java є модель динамічного завантаження класів. *Завантажувачі класів* – це спеціальні об'єкти. які забезпечують завантаження коду класів під час роботи програми.

Під час завантаження класу створюється об'єкт типу java.lang.Class. Тип Class – це клас, екземпляри якого представляють класи, інтерфейси та інші типи під час виконання Java-програми. Примітивні типи Java (boolean, byte, char, short, int, long, float і double) і void також представлені у вигляді об'єктів типу Class. Об'єкт типу Class може бути отриманий для існуючих типів (в тому числі і примітивних) з використанням ключового слова class:

1. **Які існують види завантажувачів класів?**

базовий завантажувач (bootstrap) – завантажує базові (системні) класи, які зазвичай знаходяться в jar-файлах в директорії jre/lib; базовий завантажувач не доступний програмно;  
завантажувач розширень (Extension Classloader) – завантажує класи різних пакетів розширень, які зазвичай містяться в jre/lib/ext;  
системний завантажувач (System Classloader) – реалізує стандартний алгоритм завантаження з каталогів і JAR-файлів, перелічених у змінній CLASSPATH.

1. **Як завантажити клас за ім'ям?**

String name = "SomeClass";  
Class c = Class.forName(name);

1. **Для чого створюються користувацькі завантажувачі?**

Это может быть полезно, когда нет возможности или нежелательно перечислять все используемые библиотеки при старте программы в CLASSPATH. Например, в программе должна быть возможность динамической загрузки плагинов. Или возможностей стандартного загрузчика недостаточно для загрузки нужных классов.

Абстрактний клас java.lang.ClassLoader використовується для створення типів об'єктів, що відповідають за завантаження класів. Кожен об'єкт типу Class містить посилання на ClassLoader. Для кожного класу визначений поточний завантажувач. Для того, щоб отримати завантажувач, яким був завантажений клас SomeClass, необхідно скористатися методом SomeClass.class.getClassLoader(). Java дозволяє створювати користувацькі завантажувачі.

1. **Як отримати інформацію про методи класу?**

Можна отримати список методів класу - масив об'єктів класу java.lang.reflect.Method. Цей масив буде містити посилання на відкриті методи даного класу і всіх його предків:  
  
for (int i = 0; i < o.getClass().getMethods().length; i++) {   
 System.out.println(o.getClass().getMethods()[i].getName());  
}

Для методу можна отримати інформацію про клас, в якому він оголошений (getDeclaringClass()), про ім'я (getName()), тип результату (getReturnType()). Можна також отримати масив типів параметрів (об'єктів типу Class<?>) за допомогою методу getParameterTypes(). Масив, що отримують за допомогою функції getExceptionTypes(), містить типи можливих винятків. Перевизначена функція toString() повертає повну сигнатуру методу зі списком винятків.

1. **Як викликати метод за ім'ям?**

Функція getMethod() дозволяє отримати метод за ім'ям.

Функція getDeclaredMethods() дозволяє отримати список всіх методів (відкритих, захищених, пакетних і закритих), але тільки визначених у цьому класі (а не в базових класах). Аналогічно викликають функцію getDeclaredMethod(), яка дозволяє отримати будь-який метод за ім'ям.

1. **Як викликати статичний метод?**

Метод можна викликати за допомогою функції invoke(). Перший параметр цього методу – посилання на об'єкт класу, для якого викликають метод. Для статичних методів – це null. Далі вказують необхідні фактичні параметри. Оскільки кілька методів може відрізнятися тільки списком параметрів, під час виклику getMethod()крім імені обов'язково потрібно вказувати типи параметрів (масив об'єктів типу Class). Якщо список параметрів порожній, цей масив можна опустити.  
  
Наприклад, так можна викликати статичну функцію без параметрів для класу, заданого ім'ям:  
  
method = Class.forName(className).getMethod(methodName);  
method.invoke(null);

1. **Як отримати інформацію про поля класу?**

Отримання інформації про поля здійснюється аналогічно. Клас java.lang.reflect.Field представляє дані про поле класу – імені (getName()), типу (getType()), клас, в якому визначено поле (getDeclaringClass()), модифікатори доступу (getModifiers()) тощо. Для отримання значення, яке зберігається в поле, використовують метод get(), який повертає результат типу Object. За допомогою методу set() можна встановити нове значення.

Якщо ми хочемо отримати доступ до закритих або захищених полів, але визначених саме в цьому класі, використовують методи getDeclaredField() (за ім'ям) і getDeclaredFields() (для отримання масиву полів).

1. **Як в код на Java включити код на JavaScript?**

**Засоби пакету javax.script, доданого в Java SE 6, дозволяють інтерпретувати вирази, написані на скриптових мовах (AppleScript, Groovy, JavaScript, Jelly, PHP, Python, Ruby тощо). Наприклад, таким чином можна інтерпретувати код, написаний на JavaScript:**package ua.in.iwanoff.oop.fourth;  
  
import javax.script.ScriptEngine;  
import javax.script.ScriptEngineManager;  
  
public class EvalScript {  
  
 public static void main(String[] args) throws Exception {  
 ScriptEngineManager factory = new ScriptEngineManager();  
 ScriptEngine engine = factory.getEngineByName("JavaScript");  
 engine.eval("print('Hello, World')");  
 }  
  
}

1. **Як програмно здійснити компіляцію вихідного коду?**

JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

compiler.run(null, null, null, "c:/java/test/Test.java");

1. **Дайте визначення процесу і потоку виконання.**

Процес (process) – це екземпляр комп'ютерної програми, яка завантажена в пам'ять виконується. В сучасних операційних системах процес може виконуватися паралельно з іншими процесами. Процесу виділяється окремий адресний простір, причому цей простір фізично недоступний для інших процесів.

1. **У чому переваги використання потоків виконання?**

Потоки виконання використовують для реалізації незалежних підзадач у межах одного процесу з метою реалізації фонових процесів, моделювання паралельного виконання певних дій або підвищення зручності користувацького інтерфейсу.

1. **Які є способи створення потоків виконання в Java?**

успадкування нового класу від класу java.lang.Thread і створення об'єкта нового класу;  
створення класу, який реалізує інтерфейс java.lang.Runnable; об'єкт такого класу передається конструктору класу java.lang.Thread.

1. **В яких станах може перебувати потік?**

Будь який потік виконання може знаходитися в декількох стандартних станах. Стан "новий" (Thread.State.NEW) потік отримує, коли створюється об'єкт потоку. Виклик методу start() переводить потік зі стану "новий" в стан "працює" (Thread.State.RUNNABLE). Існують стани "заблокований" (Thread.State.BLOCKED), "заблокований за часом", або "в режимі очікування" (Thread.State.TIMED\_WAITING), "очікує", або "непрацездатний" (Thread.State.WAITING) і "завершений" (Thread.State.TERMINATED). Під час створення потоку він отримує стан "новий" і не виконується. Отримати значення стану потоку можна викликом методу getState()

1. **Коли здійснюється припинення і призупинення виконання роботи потоку?**

Метод wait(long timeout), як і метод sleep(long timeout), дозволяє призупинити роботу потоку на вказану кількість мілісекунд. Під час виконання цього методу також може бути згенеровано виняток InterruptedException. Застосування цих методів дозволяє перевести потік в режим очікування (TIMED\_WAITING). На відміну від sleep(), працездатність після виклику методу wait() можна відновити методами notify() або notifyAll().

1. **Якими засобами можна здійснити синхронізацію потоків?**

Метод join() дозволяє одному потоку дочекатися завершення іншого. Виклик цього методу всередині потоку t1 для потоку t2 призведе до припинення поточного потоку (t1) до завершення t2, як показано в наведеному нижче прикладі:

1. **У яких випадках робота потоку блокується?**

У ранніх версіях Java для примусової зупинки, а також для тимчасового призупинення роботи потоку з подальшим відновленням передбачалося використання методів класу Thread - stop(), suspend() і resume().

Для забезпечення доступу до такого коду використовується концепція монітора. Під монітором розуміють деякий об'єкт, що забезпечує блокування коду під час виконання його деяким потоком. Якщо ключове слово synchronized розташоване перед заголовком функції, монітором є об'єкт, для якого викликаний цей метод (this). Після того, як викликаний хоча б один синхронізований метод, блокування поточним об'єктом поширюється на всі методи з модифікатором synchronized.

Повернути блокування об'єкта потоку можна викликом методу notify() для конкретного потоку або notifyAll() для всіх потоків. Виклик може бути здійснений тільки з іншого потоку, який заблокував, в свою чергу, зазначений об'єкт.

1. **Для чого використовується модифікатор synchronized?**

Якщо елементи даних класу оголошені як private і доступ до цієї області пам'яті можливий тільки за допомогою методів, то можна уникнути колізій, оголосивши ці методи як synchronized. Одночасно тільки один процес може викликати synchronized метод для певного об'єкта. Такий метод або фрагмент коду іменується критичної секцією.

1. **Які контейнерні класи забезпечують безпеку з точки зору потоків?**

Пакет java.util.concurrent надає набір колекцій, безпечних з точки зору багатопотоковості. Це такі узагальнені типи, як BlockingQueue (інтерфейс) і його стандартні реалізації (ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue, DelayQueue), похідні від нього інтерфейси TransferQueue (реалізація LinkedTransferQueue) і LinkedBlockingDeque (стандартна реалізація – LinkedBlockingDeque). Колекції реалізовані в пакеті java.util.concurrent.

1. **Для чого використовують клас Optional?**

Узагальнений клас Optional дозволяє зберегти значення посилання на деякий об'єкт, а також перевірити, чи встановлено значення. Наприклад, деякий метод повертає числове значення, але для деяких значень аргументу не може повернути щось певне. Такий метод може повернути об'єкт типу Optional і це значення потім може бути використано в викликає функції.

1. **Чи можна використовувати клас Optional з примітивними типами?**

yes

1. **Які додаткові можливості роботи зі стандартними контейнерами передбачені в Java 8?**

Стандартні інтерфейси пакету java.util доповнені методами, орієнтованими на використання лямбда-виразів і посилань на методи. Для забезпечення сумісності з попередніми версіями Java нові методи інтерфейсів представлені з усталеною реалізацією. Зокрема, інтерфейс Iterable визначає метод forEach(), який дозволяє виконати в циклі деякі дії, що не змінюють елементів колекції. Дію можна задати лямбда-виразом або посиланням на метод

1. **У чому переваги і особливості Stream API?**

Інтерфейс Stream розширює набір методів для роботи з потоками елементів. Це також узагальнений інтерфейс і він придатний для роботи з будь-якими посилальними типами

1. **Як отримати потік з колекції?**

Існує кілька способів створення потоку. Можна скористатися "фабричними" методами, що були додані до інтерфейсу Collection (з усталеними реалізаціями) - відповідно stream() (для послідовної роботи) и parallelStream() (для багатопотокового роботи):  
  
List<Integer> intList = Arrays.asList(3, 4, 1, 2);  
Stream<Integer> sequential = intList.stream();  
Stream<Integer> parallel = intList.parallelStream();

1. **Як отримати потік з масиву?**

Integer[] a = { 1, 2, 3 };  
Stream<Integer> fromArray = Arrays.stream(a);

1. **Чим відрізняються проміжні і кінцеві операції?**

Проміжні операції отримують і генерують потоки даних і служать для створення так званих конвеєрів (pipeline), в яких над послідовністю виконується ряд дій. Кінцеві операції дають остаточний результат і при цьому "споживають" вихідний потік. Це означає, що вихідний потік не може бути використаний повторно і в разі необхідності повинен бути створений заново.