Вопросы и ответы на собеседование по основам Java

(вторая часть)

31) В чем разница между переопределением (overriding) и перегрузкой (overloading) в Java?

OVERRIDING	OVERLOADING
1) В случае переопределения (overriding) , имена методов в суперклассе и подклассе должны быть одинаковыми.	1) В случае перегрузки (overloading) , имена методов должны быть одинаковыми.
2) Список аргументов должен быть одинаковым.	2) Список аргументов должен отличаться, по крайней мере, порядком аргументов.
3) Тип возвращаемого значения может быть таким же, или мы можем возвращать ковариантный тип. Начиная с версии 1.5, допускаются ковариантные типы.	3) Тип возвращаемого значения может быть разным в перегрузке методов.
4) Мы не можем увеличивать уровень проверяемых исключений. Нет ограничений для непроверяемых исключений.	4) В перегрузке методов могут возникать разные исключения.
5) Метод может быть переопределен только в подклассе.	5) Метод может быть перегружен в том же классе или подклассе.
6) private, static и final переменные не могут быть переопределены.	6) private, static и final переменные могут быть перегружены.

OVERRIDING	OVERLOADING
7) В переопределении, какой метод вызывается, решается во время выполнения на основе типа объекта, на который ссылается во время выполнения.	7) В перегрузке, какой метод вызывать решается на этапе компиляции на основе типа ссылки.
8) Переопределение также известно как полиморфизм времени выполнения, динамический полиморфизм или позднее связывание.	8) Перегрузка также известна как полиморфизм времени компиляции, статический полиморфизм или раннее связывание..

32). Что такое отношения между классами is A в Java?

Отношение 'является' ('**is a**') также известно как наследование. В Java мы можем реализовать отношение 'является' или наследование с помощью ключевого слова '**extends**'.
Преимущество наследования или отношения 'является' заключается в возможности повторного использования кода вместо его дублирования.
Пример: Мотоцикл является транспортным средством, автомобиль является транспортным средством. И автомобиль, и мотоцикл расширяют (extends) транспортное средство.

33). Что такое отношения между классами Has A в Java?

Отношение 'имеет' ('**has a**') также известно как '**композиция**' или '**агрегация**'. В отличие от наследования, для реализации отношения 'имеет' в Java нет специального ключевого слова. Основное преимущество отношения 'имеет' в коде на Java - это возможность повторного использования кода.

34) В чем отличие между отношениями 'IS-A' и 'HAS-A' в Java?

IS-A RELATIONSHIP	HAS- A RELATIONSHIP
1) Отношение "IS-A" в Java также известно как наследование	1) Отношение 'HAS-A' представляет агрегацию или композицию
2) Для отношения "IS-A" (является) в Java используется ключевое слово extends.	2) В Java не существует специального ключевого слова для реализации 'HAS-A' отношения, оно обычно достигается путем включения одного объекта в другой.
3) Пример: Машина (Car) - это транспортное средство (Vehicle).	3) Пример: Машина (Car) имеет двигатель (Engine). Мы не можем сказать, что Машина (Car) это двигатель (Engine).
4) Основным преимуществом наследования является возможность повторного использования кода.	4) Основным преимуществом отношения "имеет" ("HAS-A") является возможность повторного использования кода.

35). Что такое instanceof operator в Java?

Оператор **instanceof** используется для проверки типа объекта.
Синтаксис:
<выражение-ссылка> instanceof <тип-назначение>.
Оператор instanceof возвращает true, если выражение-ссылка является подтипом (или экземпляром) типа-назначения. Если выражение-ссылка равно null, то оператор instanceof вернет false.

```
public class InstanceOfExample {
public static void main(String[] args)
{Integer a = new Integer(5);
if (a instanceof java.lang.Integer) {
System.out.println(true);
} else {
System.out.println(false);
}
}
}
```

Поскольку a - это объект типа Integer, оператор instanceof вернет true. Важно отметить, что при компиляции будет проверено, является ли выражение, на которое указывает a, подтипом (подклассом) указанного типа. Если это не так, то компилятор выдаст ошибку, так как типы несовместимы.

36). Что означает null в Java?

В Java ключевое слово null используется, чтобы указать, что ссылочная переменная не указывает на какое-либо значение объекта. Например, если у вас есть переменная Employee employee;, и она не была инициализирована, то она содержит null, что означает, что она не указывает ни на какой объект.

37). Можем ли мы иметь несколько классов в одном файле?

Да, в Java можно иметь несколько классов в одном файле, но это не рекомендуется и редко используется. Вы можете объявить несколько классов в одном файле, но только один из них может быть объявлен как **public**. Если вы попытаетесь сделать два класса public в одном файле, то возникнет ошибка компиляции с сообщением "The public type must be defined in its own file" (Публичный тип должен быть объявлен в собственном файле).

38). Какие модификаторы доступа разрешены для верхнего класса?

Для верхнего уровня класса разрешены только два модификатора доступа: **public** и **default**. Если класс объявлен как public, он виден повсюду. Если класс объявлен как default, он виден только в том же пакете. Если мы попытаемся использовать модификаторы доступа private и protected для класса, мы получим следующую ошибку компиляции: "Недопустимый модификатор для класса. Для класса разрешены только public, abstract и final".

39). Что такое пакеты в Java?

Пакет - это механизм для группировки связанных классов, интерфейсов и перечислений в один модуль. Пакет можно объявить с помощью следующего оператора:

Синтаксис: **package <package-name>**

Соглашение о кодировании: имя пакета должно быть указано строчными буквами.

Оператор package определяет пространство имен. Основное назначение пакетов:

1. Решение конфликтов имен.
2. Управление видимостью: мы можем определить классы и интерфейсы, которые не доступны извне класса.

40). Можно ли иметь более одного оператора пакета в исходном файле?

Мы не можем иметь более одного оператора пакета в исходном файле. В любой программе на Java может быть, как минимум, только один оператор пакета. Если у нас есть более одного оператора пакета в исходном файле, мы получим ошибку компиляции.

41). Можно ли объявить оператор пакета после оператора импорта в Java?

Мы не можем объявить оператор пакета после оператора импорта в Java. Оператор пакета должен быть первым оператором в исходном файле. Мы можем добавлять комментарии перед оператором пакета, но не после оператора импорта.

42). Что такое идентификаторы в Java?

Идентификаторы - это имена в программе Java.
Идентификаторы могут быть именами класса, метода или переменной.
Правила для определения идентификаторов в Java:

- Идентификаторы должны начинаться с буквы, подчеркивания или знака доллара ($).
- Идентификаторы не могут начинаться с цифр.
- В идентификаторе нет ограничения на количество символов, но не рекомендуется иметь более 15 символов.
- Идентификаторы Java чувствительны к регистру.
- Первая буква может быть буквой алфавита, подчеркиванием и знаком доллара. Со второй буквы можно использовать цифры.
- Мы не должны использовать зарезервированные слова в качестве идентификаторов в Java.

43). Что такое модификаторы доступа в Java?

Важной особенностью инкапсуляции является контроль доступа. Класс, метод или переменная могут быть доступными в зависимости от модификатора доступа. В Java существует три типа модификаторов доступа: public, private, protected. Если модификатор доступа не указан, то по умолчанию используется модификатор доступа default.

44). В чем разница между спецификаторами доступа и модификаторами доступа в Java?

В C++ у нас есть спецификаторы доступа, такие как public, private, protected и default, а также модификаторы доступа, такие как static и final. Но в Java нет такого разделения на спецификаторы и модификаторы доступа. В Java у нас есть только модификаторы доступа и модификаторы, не связанные с доступом.
Модификаторы доступа в Java включают в себя:

public (публичный), private (приватный), protected (защищенный), default (по умолчанию)

Модификаторы, не связанные с доступом, включают:

abstract (абстрактный), final (финальный), static (статический), strictfp (строго точные вычисления)

45). Какие модификаторы доступа могут быть использованы для класса?

Для класса можно использовать только два модификатора доступа: public и default.

1. **public**: Класс с модификатором public виден из любого другого класса и из любого пакета.
2. **default** (или отсутствие модификатора): Если класс не имеет явно указанного модификатора доступа (то есть, не public, не private, и не protected), то он имеет модификатор доступа по умолчанию. Класс с модификатором доступа по умолчанию виден только внутри того же пакета, в котором он определен.

46). Какие модификаторы доступа можно использовать для методов?

Для методов можно использовать все модификаторы доступа: public, private, protected и default.

1. **public**: Метод с модификатором public виден из любого другого класса и из любого пакета.
2. **private**: Метод с модификатором private виден только внутри того же класса, в котором он определен. Этот метод не доступен из других классов.
3. **protected**: Метод с модификатором protected виден внутри того же пакета и в подклассах (наследниках) даже если они находятся в другом пакете.
4. **default** (или отсутствие модификатора): Если метод не имеет явно указанного модификатора доступа, то он имеет модификатор доступа по умолчанию. Метод с модификатором доступа по умолчанию виден только внутри того же пакета, в котором он определен.

47). Какие модификаторы доступа можно использовать для переменных?

Для переменных можно использовать все модификаторы доступа: public, private, protected и default.

1. **public**: Переменная с модификатором public видна из любого другого класса и из любого пакета.
2. **private**: Переменная с модификатором private видна только внутри того же класса, в котором она определена. Эта переменная не доступна из других классов.
3. **protected**: Переменная с модификатором protected видна внутри того же пакета и в подклассах (наследниках) даже если они находятся в другом пакете.
4. **default** (или отсутствие модификатора): Если переменная не имеет явно указанного модификатора доступа, то она имеет модификатор доступа по умолчанию. Переменная с модификатором доступа по умолчанию видна только внутри того же пакета, в котором она определена.

48). Что такое модификатор доступа final в Java?

Модификатор доступа final в Java используется для указания, что элемент (переменная, метод или класс) является "завершенным" и не может быть изменен или наследован в дальнейшем. Вот, как final применяется к разным элементам:

Переменные: Переменная с модификатором final является константой и не может быть изменена после ее инициализации. Попытка изменения значения final переменной приведет к ошибке компиляции.

Методы: Метод с модификатором final не может быть переопределен (перекрыт) в подклассах. Это используется, чтобы предотвратить изменение реализации метода в наследниках.

Классы: Класс с модификатором final не может быть расширен (наследоваться). Такой класс считается "завершенным", и нельзя создавать подклассы.

49). Объяснение абстрактных классов в Java:

Signature :
abstract class <class-name>
{
}

Иногда возникают ситуации, когда невозможно предоставить реализацию для всех методов в классе. Мы хотим оставить реализацию для класса, который наследует его. В таком случае мы объявляем класс абстрактным.
Для создания абстрактного класса мы используем ключевое слово **abstract**.
Любой класс, содержащий один или несколько абстрактных методов, объявляется как абстрактный. Если мы не объявляем класс как абстрактный, который содержит абстрактные методы, мы получаем ошибку времени компиляции с сообщением "Тип <имя-класса> должен быть абстрактным классом для определения абстрактных методов."
Пример: если мы возьмем класс Vehicle, мы не можем предоставить реализацию для него, потому что могут существовать двухколесные, четырехколесные и другие виды транспорта. В таком случае мы делаем класс Vehicle абстрактным. Все общие характеристики транспортных средств объявляются как абстрактные методы в классе Vehicle. Любой класс, который наследует Vehicle, предоставит свою собственную реализацию методов. Это ответственность подкласса предоставить реализацию.

49). продолжение

Основные характеристики абстрактных классов:

1. Абстрактные классы не могут быть инстанциированы.
2. Абстрактные классы могут содержать абстрактные методы, конкретные методы или оба вида методов.
3. Любой класс, который наследует абстрактный класс, должен переопределить все абстрактные методы абстрактного класса.
4. Абстрактный класс может содержать либо 0, либо несколько абстрактных методов.

Хотя мы не можем создавать экземпляры абстрактных классов, мы можем создавать ссылки на объекты и у абстрактного класса может быть конструктор , который можно вызвать через ключевое слово **super** . С помощью ссылок на суперклассы мы можем ссылаться на подклассы.

50). Что такое абстрактный метод в Java?

Абстрактный метод - это метод, у которого нет тела.
Абстрактный метод объявляется с ключевым словом abstract и точкой с запятой вместо тела метода.

Сигнатура: public abstract void <имя метода>();
Пример: public abstract void getDetails();

Обязанность предоставить реализацию абстрактного метода, объявленного в абстрактном классе, лежит на наследующем классе.

51). Что такое исключение в Java?

В Java исключение - это объект.
Исключения создаются, когда в нашей программе возникают аномальные ситуации.
Исключения могут быть созданы как виртуальной машиной Java (JVM), так и нашим кодом приложения.
Все классы исключений определены в пакете **java.lang**.
Иными словами, исключение можно назвать ошибкой времени выполнения.

52). Укажите некоторые ситуации, когда в Java могут возникнуть исключения?

1. Попытка доступа к элементу, который не существует в массиве.
2. Некорректное преобразование числа в строку и строки в число (NumberFormatException).
3. Некорректное приведение типов классов (ClassCastException).
4. Попытка создания объекта для интерфейса или абстрактного класса (InstantiationException).

53). Что такое обработка исключений в Java?

Обработка исключений - это механизм, определяющий, как поступать в случае возникновения аномальной ситуации в программе. Когда исключение возникает в программе и не обрабатывается должным образом, это может привести к завершению программы. Значение обработки исключений заключается в том, чтобы избежать внезапного завершения программы и продолжить выполнение остальной части программы нормально. Это можно сделать с помощью механизма обработки исключений.

54). Что такое Error в Java?

Ошибка (Error) - это подкласс класса **Throwable** в Java. Когда ошибки вызываются нашей программой, мы называем это исключением (Exception), но иногда исключения возникают из-за проблем с окружением, такими как исчерпание памяти. В таких случаях мы не можем обработать исключения. Исключения, которые не могут быть восстановлены, называются ошибками (Errors) в Java.
Пример: проблемы с исчерпанием памяти (Out of memory errors).

55). Каковы преимущества обработки исключений в Java?

1. Разделение обычного кода и кода обработки исключений, чтобы избежать аномального завершения программы.
2. Категоризация разных типов исключений, позволяя обрабатывать конкретные исключения вместо всех сразу с использованием корневого класса Exception. Рекомендуется обрабатывать исключения конкретными исключениями, а не общим классом Exception.
3. Механизм стека вызовов: если метод генерирует исключение, и оно не обрабатывается немедленно, то исключение передается или выбрасывается вызывающему методу. Это продолжается до тех пор, пока не будет найден соответствующий обработчик исключений. Если обработчик не будет найден, программа завершится аварийно.

56). Сколькими способами мы можем обрабатывать исключения в Java?

Мы можем обрабатывать исключения двумя способами в Java:

1. Путем указания блока try-catch, в котором мы можем перехватывать исключение.
2. Объявление метода с ключевым словом throws.

57). Назовите пять ключевых слов, связанных с обработкой исключений.

Список пяти ключевых слов, связанных с обработкой исключений:

1. Try
2. Catch
3. Throw
4. Throws
5. Finally

58). Объясните, try и catch ключевые слова в Java?

Ключевое слово **try** используется для определения блока кода, который может вызвать исключение. В блоке try мы помещаем код, который потенциально может привести к возникновению исключения.
Ключевое слово **catch** используется для обработки исключений, которые могут быть сгенерированы в блоке try. В блоке catch указывается тип исключения, которое мы хотим перехватить, и код, который должен выполняться в случае возникновения такого исключения.
Совместное использование ключевых слов try и catch позволяет нам обработать исключения в нашей программе, предотвратив ее некорректное завершение из-за ошибок.

Syntax :

```
try
{
}
Catch(Exception e) {
{
}
```

59). Можем ли мы использовать блок try без блока catch?

В Java каждый блок try должен иметь хотя бы один блок catch или finally. Если блок try не имеет ни блока catch, ни блока finally, то это вызовет ошибку компиляции. Мы можем пропустить либо блок catch, либо блок finally, но не оба одновременно.

60). Можем ли мы иметь несколько блоков catch для блока try?

Да, в Java можно иметь несколько блоков catch для одного блока try. В некоторых случаях ваш код может генерировать более одного исключения, и в таком случае можно указать два или более блока catch, каждый из которых обрабатывает разные типы исключений. Когда исключение возникает, JVM проверяет каждый блок catch в порядке и выполняет первый, который соответствует типу исключения, после чего остальные блоки catch пропускаются.

Важно отметить, что порядок блоков catch имеет значение, и они должны следовать от наиболее конкретных типов исключений к более общим.