Вопросы по теме Класс Object:

1. Почему все классы наследуются от класса Object?
2. Назовите методы класса Object?
3. Можно ли создать экземпляр класса Object?
4. Зачем кому-то может понадобится создавать экземпляр класса Object?
5. В чем разница между a.getClass() и A.class в Java?
6. Что делает метод finalize()?
7. Что будет, если finalize() подвиснет или выбросит исключение?
8. Чем отличаются final, finally, finalize()?

Класс Object – базовый класс в языке Java. От него неявно наследуются все классы.

Назовите методы класса Object.

equals() hashCode() clone() finalize() getClass() notify() notifyAll() wait() toString()

Почему все классы наследуются от класса Object.

Все классы в Java неявно наследуются от класса Object. Это означает, что каждый класс в Java обладает теми же методами, что и класс Object. Это позволяет:

– существовать структурам, работа которых основана на работе метода *hashCode()*, например, HashMap и HashSet;

– сравнивать все объекты между собой с помощью методов *equals()* и *hashCode()*;

– представить любой объект в строковом виде с помощью *toString()*;

– вызывать сборщик мусора с помощью *finalize()*, чтобы объекты без ссылок не занимали память;

– работать с любым объектом в рамках многопоточности с помощью методов *notify()*, *notifyAll()*, *wait()*;

– создавать копии объектов с помощью метода *clone()*.

Также мы можем использовать класс Object в рамках полиморфизма:

void someMethod (Object obj) { } // можно передать в метод объект любого класса

Можно ли создать экземпляр класса Object и зачем это нужно.

Можно. Object lock = new Object();

Может пригодится, если хотим выполнить синхронизацию, когда this используемого объекта уже используется для блокировки какого-либо состояния этого же объекта. Или если не хотим, чтобы кто-то мог использовать блокировку.

```
void someMethod() {
        synchronized(lock) {
                // что-то делаем, блокировка метода «висит» на объекте lock
        }
}
```

В чем разница между a.getClass() и A.class в Java.

A.class работает во время компиляции

a.getClass() работает во время выполнения программы и требует экземпляр объекта

class Child extends Parent

Parent parent = new Child();

System.out.println(parent.getClass()); // вывод на экран: class Child

System.out.println(Parent.class); // вывод на экран: class Parent

Что делает метод finalize()?

Метод finalize() вызывается Java-машиной у объекта перед тем, как объект будет уничтожен сборщиком мусора (Garbage Collector). Теоретически, переопределив этот метод, можно освобождать ресурсы перед уничтожением объекта, но на практике так лучше не поступать. Мы не знаем, когда JVM даст команду сборщику мусора удалить неиспользуемый объект. Этого может вообще не произойти. Также переопределение метода finalize() значительно удлиняет время жизни объекта после смерти (из-за принципа работы сборщика мусора). На данный момент метод в официально документации значится как «Deprecated», т.е. пользоваться этим методом без особой надобности не стоит. Освобождать ресурсы лучше, когда объект еще жив, и вы будете уверены, что ресурсы будут освобождены.

Явно вызвав метод finalize() мы не запустим сборщик мусора, это работает как подсказка для JVM, что в этом месте нужно почистить память. Но и этот момент лишний, т.к. JVM лучше знает, когда и где нужно вызвать сборщик мусора.

Что будет, если finalize() подвиснет или выбросит исключение?

Сборщик мусора не вызывает методы finalize() напрямую, а только добавляет соответствующие объекты с помощью вызова метода java.lang.ref.Finalizer.register(Object) в специальный двусвязный список. Этот список хранит в себе ссылки на объекты, для которых необходимо вызвать finalize().

Непосредственный вызов методов finalize() происходит в отдельном потоке (java.lang.ref.Finalizer.FinalizerThread), который создаётся при запуске JVM. Методы finalize() вызываются последовательно в том порядке, в котором были добавлены в список сборщиком мусора.

1. Если в методе finalize() возникнет ошибка, то она будет проигнорирована.
2. Если метод finalize() подвиснет, то подвиснет поток, но не сам сборщик мусора.

Таким образом, если объект не имеет метода finalize(), то он будут удаляться, а объекты с методом finalize() будут увеличивать очередь (двусвязный список) до тех пор, пока не отвиснет поток FinalizerThread, не завершится приложение или не кончится память.

Чем отличаются final, finally, finalize()?

Ключевое слово **final** имеет функционал в зависимости от того, к чему применяется:

– класс не может иметь наследников;

– метод не может быть переопределен в классах наследниках;

– для переменных ссылочного типа означает, что после присвоения переменной объекта, нельзя изменить ссылку на данный объект;

– для переменных примитивного типа означает, что однажды присвоенное значение не может быть изменено;

– параметр метода не может менять свое значение внутри метода.

Оператор **finally** гарантирует выполнение кода в блоке try-catch независимо от вызванного исключения.

Метод **finalize()** вызывается Java-машиной у объекта перед тем, как объект будет уничтожен сборщиком мусора.