**Класс Object**

1) Почему все объекты наследуются от Object  
2)Можно ли создавать объект Obj и для чего это может быть нужно  
3) Методы Obj  
4) Методы vait notify notifyAll  
5) Что такое HashCode, что он возвращает по умолчанию  
6) Связь equals и HashCode  
7) Метод finalize

1) Как и любое другое применение наследования для того, чтобы получить открытый функционал базового класса (методы базового класса гарантированно будут во всех классах) и получение возможности использовать дочерний класс в любом месте, где требуется объект базового класса (это скорее, как побочный эффект).

2) Можно создать. Может пригодится, если необходим только базовый функционал, например, использование объекта Object в качестве монитора или создание контейнера объектов, тип которых неважен.

3) Часто используемые - equals, hashCode, toString  
Методы многопоточности - wait, notify, notifyAll   
Метод рефлексии - getClass   
Клонирования - clone   
finalize

4) Методы, использующиеся в synchronized блоках или методах, работают для синхронизации потоков, назвать что делают;

5 / 6) **HashCode и equials**

**Equilas** – Метод, необходимый для логического подтверждения или отрицания того, что объекты являются равными. Базовая реализация сравнивает ссылки на объекты, поэтому для пользовательского класса необходимо переопределение для корректности выходных данных.

*Нет необходимости переопределять если:*

Объекты этого класса являются уникальными или класс подразумевает поведение, а не работу с данными – например Thread.  
Уже имеет свою реализацию и ее поведение является приемлемым, например, Set, List, Map.  
Когда область видимости вашего класса private или package-private и метод никогда не будет вызван.

*Контракт equals:*

**Рефлексивность** - x.equals(x) ==  true. (Объект должен быть равен сам себе)  
**Симметричность** – если x.equals(y)==true, то y.equals(x) == true (Последовательность аргументов не влияет на результат)  
**Транзитивность** – если x.equals(y) == true и x.equals(z) == true, то z.equals(y) == true   
**Согласованность** – повторный вызов equals с теми же аргументами должен давать одинаковый результат.  
**Сравнение** **null (Определенность)** - x.equals(null) == false

\*Вещественные числа нельзя сравнивать через == из-за погрешности, использовать compare()

**HashCode** – число, полученное в результате хэширования. Хэширование – алгоритм преобразование данных объекта любой длинны в данные заданной длины. В Java возвращает int.

Требования для hash-функции:

Быстрота – обращение к этой функции — это частая операция, поэтому должна быть быстрой.

Необратимость - если берется hash из 123 и он дает 2, нельзя из 2х получить 123

Распределенность – функция должна обеспечивать наименьшую коллизию.

Фиксированный размер – всегда возвращать фиксированный размер числа, независимо от входных данных.

**Контракт hashCode:**

Повторный вызов hashcode в течение жизни приложения должен возвращать тот же hashcode, при условии, что поля, участвующие в вычислении, не были изменены.

Равные по equals объекты должны иметь одинаковый hashcode.

Разные по equals объекты могут иметь одинаковый хэш код, но функция должна стремиться к уменьшению таких ситуаций.

**Стандартная реализация hashcode:**

Если реализация hashcode не переопределена, то в качестве hashcode возвращается идентификационных хэш (identity hash code), он вычисляется один раз, независимо от того, переопределен hashcode или нет (используется для нужд JVM) хранится в заголовке объекта в слове mark.

Реализации identityHashCode – в HotSpot можно выбирать реализацию Object.hashCode с помощью ключа командной строки -XX:hashCode=n (где n от 0 до 5). Значение получается с помощью System.identityHashCode().

  0 – Используется генератор псевдослучайных чисел  
  1 – Функция адреса объекта в памяти с учетом их возможного перемещения  
  2 – константа 1 – всегда возвращает 1  
  3 – последовательный счетчик  
  4 – адрес объекта без модификаций  
  5 – алгоритм основанный на локальном состоянии потока + псевдослучайные числа.

\*Информация про identity hash и его реализации актуальна для HotSpot\*

Хэш массивов вычисляется на основе его наполнения, а именно рекурсивно вызывается метод hashcode для каждого элемента массива и для лучшего распределения hash code результат каждого вычисленного hash code умножается на простое число 31 (простые числа дают лучшее распределение).

public int hashCode() {  
   
 return hash(age, name);  
}  
public int hash(Object... values) {   
 if (values == null) return 0;  
 int result = 1;  
 for (Object elem : values) {  
 result = 31 \* result + (elem == null ? 0 : elem.hashCode());  
 }  
 return result;  
}

**Переопределение equals и hashcode вместе обязательно по причине того, что классы или методы, которые опираются на правила этого контракта могут работать некорректно. (Hash коллекции).**

7) Выполняет действия по очистке ресурсов, которые использует объект перед окончательным удалением. Метод имеет несколько проблем и ограничений и не рекомендуется к использованию (deprecated):

Неопределенность времени вызова – когда вызывать метод решает GC (перед очисткой), а удаление объекта предсказать проблематично.  
Проблемы с производительностью.  
Игнорирование исключений .

1. finalize() можно использовать только в двух случаях:  
1.1. Проверка/подчистка ресурсов с логированием  
1.2. При работе с нативным кодом, который не критичен к утечке ресурсов  
2. finalize() замедляет работу GC по очистке объекта в 430 раз  
3. finalize() может быть не вызван