методу во время компиляции. И заметьте, ошибки во время компиляции не возникло!

5. Объявить статическим также можно и класс, за исключением классов верхнего уровня. Такие классы известны как «вложенные статические классы» (`nested static class`). Они бывают полезными для представления улучшенных связей. Яркий пример вложенного статического класса — `HashMap.Entry`, который предоставляет структуру данных внутри `HashMap`. Стоит заметить, также как и любой другой внутренний класс, вложенные классы находятся в отдельном файле .class. Таким образом, если вы объявили пять вложенных классов в вашем главном классе, у вас будет 6 файлов с расширением .class. Ещё одним примером использования является объявление собственного компаратора (`Comparator`), например компаратор по возрасту (`AgeComparator`) в классе сотрудники (`Employee`).

6. Модификатор `static` также может быть объявлен в статичном блоке, более известным как «Статический блок инициализации» (`Static initializer block`), который будет выполнен во время загрузки класса. Если вы не объявите такой блок, то Java соберёт все статические поля в один список и выполнит его во время загрузки класса. Однако, статичный блок НЕ может пробросить перехваченные исключения, но может выбросить не перехваченные. В таком случае возникнет «Exception Initializer Error». На практике, любое исключение возникшее во время выполнения и инициализации статических полей, будет завёрнуто Java в эту ошибку. Это также самая частая причина ошибки «No Class Def Found Error», т.к. класс не находился в памяти во время обращения к нему.

7. Полезно знать, что статические методы связываются во время компиляции, в отличие от связывания виртуальных или не статических методов, которые связываются во время исполнения на реальном объекте. Следовательно, статические методы не могут быть переопределены в Java, т.к. полиморфизм во время выполнения не распространяется на них. Это важное ограничение, которое необходимо учитывать, объявляя метод статическим. В этом есть смысл, только тогда, когда нет возможности или необходимости переопределения такого метода классами-наследниками. Методы-фабрики и методы-утилиты хорошие образцы применения модификатора `static`. Джошуа Блох выделил несколько преимуществ использования статичного метода-фабрики перед конструктором, в книге «Effective Java», которая является обязательной для прочтения каждым программистом данного языка.

8. Важным свойством статического блока является инициализация. Статические поля или переменные инициализируются после загрузки класса в память. Порядок инициализации сверху вниз, в том же порядке, в каком они описаны в исходном файле Java класса. Поскольку статические поля инициализируются на потокобезопасный манер, это свойство также используется для реализации паттерна `Singleton`. Если вы не используется список `Enum` как `Singleton`, по тем или иным причинам, то для вас есть хорошая альтернатива. Но в таком случае необходимо учесть, что это не «ленивая» инициализация. Это означает, что статическое поле будет проинициализировано ещё ДО того как кто-нибудь об этом «попросит». Если объект ресурсоёмкий или редко используется, то инициализация его в статическом блоке сыграет не в вашу пользу.

9. Во время сериализации, также как и `transient` переменные, статические поля не сериализуются. Действительно, если сохранить любые данные в статическом поле, то после десериализации новый объект будет содержать его первичное (по-умолчанию) значение, например, если статическим полем была переменная типа `int`, то её значение после десериализации будет равно нулю, если типа `float` – 0.0, если типа `Object` – `null`. Честно говоря, это один из наиболее часто задаваемых вопросов касательно сериализации на собеседованиях по Java. Не храните наиболее важные данные об объекте в статическом поле!

10. И напоследок, поговорим о `static import`. Данный модификатор имеет много общего со стандартным оператором `import`, но в отличие от него позволяет импортировать один или все статические члены класса. При импортировании статических методов, к ним можно обращаться как будто они определены в этом же классе, аналогично при импортировании полей, мы можем получить доступ без указания имени класса. Данная возможность появилась в Java версии 1.5, и при должном использовании улучшает читабельность кода. Наиболее часто данная конструкция встречается в тестах *JUnit*, т.к. почти все разработчики тестов используют `static import` для assert методов, например `assertEquals()` и для их перегруженных дубликатов. Если ничего не понятно – добро пожаловать за дополнительной информацией.

На этом всё. Все вышеперечисленные пункты о модификаторе *static* в Java обязан знать каждый программист. В данной статье была рассмотрена базовая информация о статических переменных, полях, методах, блоках инициализации и импорте. В том числе некоторые важные свойства, знание которых является критичным при написании и понимании программ на Java. Я надеюсь, что каждый разработчик доведёт свои навыки использования статических концептов до совершенства, т.к. это очень важно для серьёзного программирования."