1. **вопросов по Java:**
2. *Что такое ООП?*

ООП – методология программирования, основанная на представлении программного продукта в виде совокупности объектов, каждый из которых является экземпляром конкретного класса. ООП использует в качестве базовых элементов эвристическое взаимодействие объектов.

1. *Что такое объект?*

Объект – реальная именованная сущность, обладающая свойствами и проявляющая свое поведение.

Объект – обладающий именем набор данных (полей объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним. Имя используется для доступа к полям и методам, составляющим объект. В предельных случаях объект может не содержать полей или методов, а также не иметь имени.

1. *Что такое класс?*

Класс-шаблон или абстракция сущности предметной области, которая содержит описание данных и операций над объектами.

1. *Назовите 3 основных принципы ООП.*

Инкапсуляция – сокрытие внутренней реализации поведения класса от внешней среды. Внешней среде доступны только вызовы методов (интерфейс класса).

Наследование – возможность создания нового класса на основе существующего.

Полиморфизм – возможность использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

1. *Что такое наследование?*

Наследование – возможность создания нового класса на основе существующего.

Один класс может наследовать или расширять поля и методы другого класса с помощью ключевого слова extends. Класс, который выступает базой для расширения, – «суперкласс», класс, который непосредственно проводит расширение, - подкласс. Подкласс имеет доступ ко всем открытым и защищенным полям и методам суперкласса, так, словно они описаны в подклассе: производный класс не имеет доступа к закрытым полям и методам класса. Также подкласс может объявлять методы и переопределять методы.

1. *Что такое полиморфизм? Какие проявления полиморфизма в Java Вы знаете?*

Полиморфизм – возможность применения одноименных методов с одинаковыми или различными наборами параметров в одном классе или в группе классов, связанных отношением наследования.

Проявления: переопределение, перегрузка и приведение типов (На основе описания классов компилятор проверяет, сужает или расширяет возможности класса программист, объявляющий переменную. Если переменной суперкласса присваивается объект подкласса, возможности класса сужаются, и компилятор без проблем позволяет программисту сделать это. Если, наоборот, объект суперкласса присваивается переменной подкласса расширяются, поэтому программист должен подтвердить это с помощью обозначения, предназначенного для приведения типов, указав в скобках имя подкласса (subclass)).

1. *Перегрузка. Разница между перегрузкой и переопределением.*

Переопределение метода — одна из возможностей языка программирования, позволяющая подклассу или дочернему классу обеспечивать специфическую реализацию метода, уже реализованного в одном из суперклассов или родительских классов. Реализация метода в подклассе переопределяет (заменяет) его реализацию в суперклассе, описывая метод с тем же названием, что и у метода суперкласса, а также у нового метода подкласса должны быть те же параметры или сигнатура, тип возвращаемого результата, что и у метода родительского класса. Версия метода, которая будет исполняться, определяется объектом, используемым для его вызова. Если вызов метода происходит от объекта родительского класса, то выполняется версия метода родительского класса, если же объект подкласса вызывает метод, то выполняется версия дочернего класса. //// Переопределенный метод – метод ,описанный в производном классе, сигнатура этого метода совпадает с сигнатурой метода, описанного в суперклассе.

Перегрузка метода заключается в следующем — вы создаете метод с таким же именем, но с другим набором параметров, методы – перегруженные. Перегруженные методы различаются по типу и/или количеству параметров. Возвращаемые типы перегруженных методов могут быть различны, самого возвращаемого типа не достаточно для различения 2-х версий метода. Когда Java встречает вызов перегруженного метода, она просто выполняет ту версию, параметры которой соответствуют аргументам, использованным в вызове.

1. *Что такое инкапсуляция?*

Инкапсуляция означает, что данные объекта недоступны непосредственно, а их можно получить или изменить используя МЕТОДЫ этого класса. Они инкапсулируются — скрываются от прямого доступа извне. Инкапсуляция предохраняет данные объекта от нежелательного доступа, позволяя объекту самому управлять доступом к своим данным.

1. *Чем отличается JRE, JVM и JDK?*

**JVM:**

Виртуальная машина Java (JVM) - это виртуальная машина, в которой выполняется твой скомпилированный байт-коды Java. JVM не понимает Java typo, поэтому вы компилируете свои \*.java файлы для получения файлов \*.class, содержащих байт-коды, понятные JVM. Это также объект, который позволяет Java быть "переносимым языком" (писать один раз, работать в любом месте). Действительно, существуют конкретные реализации JVM для разных систем (Windows, Linux, MacOS, см. Список википедий.), Цель состоит в том, что с одинаковыми байткодами они дают одинаковые результаты.

**Java Runtime Environment (JRE)**

Java Runtime Environment (JRE) предоставляет библиотеки, виртуальную машину Java и другие компоненты для запуска апплетов и приложений, написанных на языке программирования Java. JRE не содержит инструментов и утилит, таких как компиляторы или отладчики для разработки апплетов и приложений.

**Java Development Kit (JDK)**

JDK(Java Development Kit) —  стандартная версия платформы Java. JDK предназначена для разработки. JDK —  это специальный пакет разработчика, включающий в себя документацию, различные библиотеки классов, утилиты, документацию, компилятор а также саму исполнительную систему JRE.

1. *Опишите модификаторы доступа в Java*.

Модификаторы доступа можно рассматривать как с позиции инкапсуляции так и наследования. Если рассматривать с позиции инкапсуляции, то модификаторы доступа позволяют ограничить нежелательный доступ к членам класса извне.

Член класса (поле или метод), объявленный **public**, доступен из любого места вне класса.

Все, что объявлено **private**, доступно только методам внутри класса и нигде больше.

Если у элемента вообще не указан модификатор уровня доступа, то такой элемент будет виден и доступен из подклассов и классов того же пакета. Именно такой уровень доступа используется по умолчанию.

Если же необходимо, чтобы элемент был доступен из другого пакета, но только подклассам того класса, которому он принадлежит, нужно объявить такой элемент со спецификатором **protected**. Действие спецификаторов доступа распространяется только на тот элемент класса, перед которым они стоят.

Спецификатор доступа **public** может также стоять перед определением внешнего (enclosing) класса. Если данный спецификатор отсутствует, то класс недоступен из других пакетов.

1. *Абстрактный класс*

Абстрактные классы (asbtract) содержат объявления абстрактных методов, которые не реализованы в этих классах, а будут реализованы в подклассах. Объекты таких классов создать нельзя, но можно создать объекты подклассов, которые реализуют эти методы. При этом допустимо объявлять ссылку на абстрактный класс, но инициализировать ее можно только с объектом производного от него класса. Абстрактные классы могут содержать и полностью реализованные методы, а также конструкторы и поля данных.

**Нельзя использовать конструктор абстрактного класса для создания его объекта.** Кроме обычных методов **абстрактный класс может содержать абстрактные методы.** Такие методы определяются с помощью ключевого слова abstract и не имеют никакого функционала:

public abstract void display();

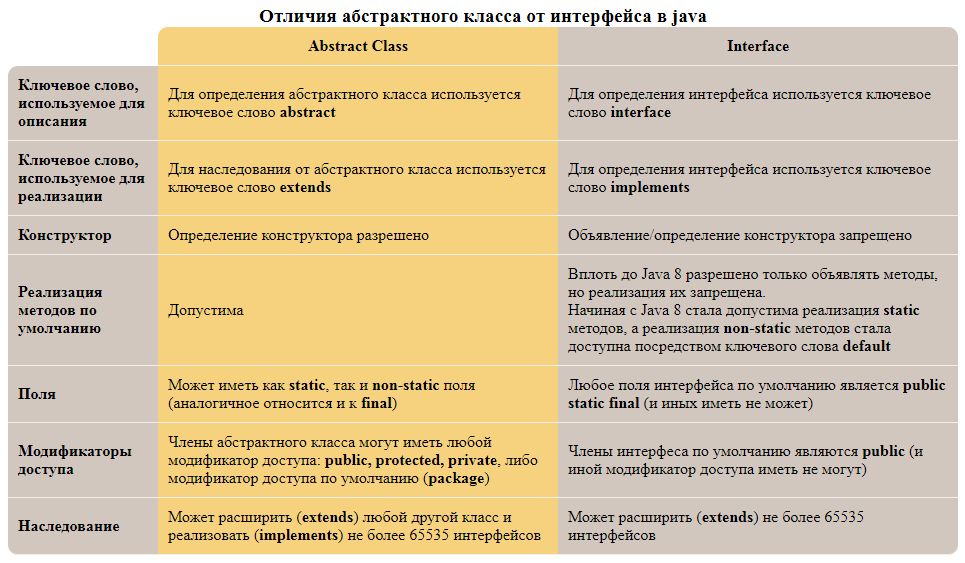
**Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе.** Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

1. *Интерфейс. Отличия абстрактного класса и интерфейса? В каких случаях Вы бы использовали абстрактный класс, а в каких интерфейс?*

Интерфейс – описание тех методов, которые должен реализовывать класс, имплементирующий данный интерфейс. Интерфейсы применяются для добавления к классам новых возможностей, которых нет и не может быть в базовых классах. Интерфейсы говорят о том, что класс может сделать, но не говорят, как он должен это сделать. Интерфейс только гарантирует (определяет контракт), какие методы должен выполнять класс, но как класс выполняет контракт, интерфейс контролировать не может.

Поля интерфейса по умолчанию являются final static. Все методы по умолчанию открыты (public).

С абстрактными классами вы теряете индивидуальность класса, наследующего его; с интерфейсами вы просто расширяете функциональность каждого класса.



1. *Память в Java?*

Память в java условно можно разделить на 2 части: stack (стек) и heap (куча).

**Java Heap (куча)** используется Java Runtime для выделения памяти под объекты и JRE классы. Создание нового объекта также происходит в куче. Здесь работает сборщик мусора: освобождает память путем удаления объектов, на которые нет каких-либо ссылок. Любой объект, созданный в куче, имеет глобальный доступ и на него могут ссылаться из любой части приложения.

**Stack память в Java**

Стековая память в Java работает по схеме LIFO (Последний-зашел-Первый-вышел). Всякий раз, когда вызывается метод, в памяти стека создается новый блок, который содержит примитивы и ссылки на другие объекты в методе. Как только метод заканчивает работу, блок также перестает использоваться, тем самым предоставляя доступ для следующего метода.

Размер стековой памяти намного меньше объема памяти в куче.

1. *Работа “кучи”?*

• **Eden Space (heap)** – в этой области выделятся память под все создаваемые из программы объекты. Большая часть объектов живет недолго (итераторы, временные объекты, используемые внутри методов и т.п.), и удаляются при выполнении сборок мусора это области памяти, не перемещаются в другие области памяти. Когда данная область заполняется (т.е. количество выделенной памяти в этой области превышает некоторый заданный процент), GC выполняет быструю (minor collection) сборку мусора. По сравнению с полной сборкой мусора она занимает мало времени, и затрагивает только эту область памяти — очищает от устаревших объектов Eden Space и перемещает выжившие объекты в следующую область.   
• **Survivor Space (heap)** – сюда перемещаются объекты из предыдущей, после того, как они пережили хотя бы одну сборку мусора. Время от времени долгоживущие объекты из этой области перемещаются в Tenured Space.

**Tenured (Old) Generation (heap)** — Здесь скапливаются долгоживущие объекты (крупные высокоуровневые объекты, синглтоны, менеджеры ресурсов и проч.). Когда заполняется эта область, выполняется полная сборка мусора (full, major collection), которая обрабатывает все созданные JVM объекты.  
• **Permanent Generation (non-heap)** – Здесь хранится метаинформация, используемая JVM (используемые классы, методы и т.п.). В частноси **Code Cache (non-heap),** где кешируется скомпилированный платформенно — зависимый код.

1. *Как работает сборщик мусора (garbage collector)?*

Java-программисту не нужно следить за распределением памяти, так как **сборщик мусора** управляет памятью автоматически. Сборщик мусора запускается виртуальной машиной Java (JVM). Сборщик мусора - это низкоприоритетный процесс, который запускается периодически и освобождает память, использованную объектами, которые больше не нужны.

JVM обычно запускает сборщик мусора при низком уровне свободной памяти. Но работа сборщика мусора не гарантирует, что всегда будет оставаться достаточно свободной памяти.

Вы можете запросить запуск сборщика мусора, но вы не можете принудительно задавать это действие.

Для запроса можно вызвать один из следующих методов:

**- System.gc()**

**- Runtime.getRuntime().gc()**

При удалении из памяти объектов, некоторым аналогом является метод finalize(),

в тело которого помещается код по освобождению занятых объектом ресурсов.

Виртуальная машина станет вызывать его каждый раз, когда сборщик мусора

будет уничтожать объект класса, которому не соответствует ни одна ссылка.

1. *Расскажите про приведение типов.  Что такое понижение и повышение  типа?*

Приведение типов: (На основе описания классов компилятор проверяет, сужает или расширяет возможности класса программист, объявляющий переменную. Если переменной суперкласса присваивается объект подкласса, возможности класса сужаются, и компилятор без проблем позволяет программисту сделать это. Если, наоборот, объект суперкласса присваивается переменной подкласса расширяются, поэтому программист должен подтвердить это с помощью обозначения, предназначенного для приведения типов, указав в скобках имя подкласса (subclass)).

1. *Какие методы есть у класса Object?*

Object это базовый класс для всех остальных объектов в Java**.**Каждый класс наследуется от Object. Соответственно все классы наследуют методы класса Object**.**Методы класса Object: public final native Class **getClass**()

* public native int **hashCode**()
* public boolean **equals**(Object obj)
* protected native Object **clone**() throws CloneNotSupportedException
* public String **toString**()
* protected void **finalize**() throws Throwable

1. *Метод hashCode и equals.*

Equals и hashCode являются фундаментальными методами объявленные в классе Object и содержатся в стандартных библиотеках Java. Метод еquals() используется для сравнения объектов, а hashCode - для генерации целочисленного кода объекта. Реализация по умолчанию метода equals() в классе java.lang.Object сравнивает ссылки на адреса в памяти, которые хранят переменные, и возвращает true только в том случае, если адреса совпадают, другими словами переменные ссылаются на один и тот же объект. Java рекомендует переопределять методы equals() и hashCode(), если предполагается, что сравнение должно осуществляться в соответсвии с естественной логикой или бизнес-логикой. Многие классы в стандартных библиотеках Java переопределяет их, например в классе String переопределяется equals таким образом, что возвращается true, если содержимое двух сравниваемых объектов одинаковое.

Java предлагает следующие правила для переопределения этих методов:

- Рефлексивность: Объект должен равняться себе самому.

- Симметричность: если a.equals(b) возвращает true, то b.equals(a) должен тоже вернуть true.

- Транзитивность: если a.equals(b) возвращает true и b.equals(c) тоже возвращает true, то c.equals(c) тоже должен возвращать true.

- Согласованность: повторный вызов метода equals() должен возвращать одно и тоже значение до тех пор, пока какое-либо значение свойств объекта не будет изменено. То есть, если два объекта равны в Java, то они будут равны пока их свойства остаются неизменными.

- Сравнение null: объект должны быть проверен на null. Если объект равен null, то метод должен вернуть false, а не NullPointerException. Например, a.equals(null) должен вернуть false.

**Соглашение между equals и hashCode в Java.**

1. Если объекты равны по результатам выполнения метода equals, тогда их hashcode должны быть одинаковыми.
2. Если объекты не равны по результатам выполнения метода equals, тогда их hashcode могут быть как одинаковыми, так и разными. Однако для повышения производительности, лучше, чтобы разные объекты возвращали разные коды.
3. *Метод ToString.*

Этот метод служит для представления объекта в виде строки. Это требуется например если мы хотим вывести объект на экран.

Метод **toString()** есть у всех объектов и все объекты используют этот метод при работе со строками. По-умолчанию он состоит из двух составляющих разделенных собачкой. Эти составляющие: **имя\_класса\_объекта** и **хэш\_кода**.

1. *Что такое конструктор. Виды конструктора?*

Конструктор- метод, который автоматически вызывается при создании объекта класса и выполняет действия только по инициализации объекта. По сути, конструктор нужен для автоматической инициализации переменных.

Конструктор:

-имеет то же имя, что и класс

-вызывается не по имени, а только вместе с ключевым словом New при создании экземпляра класса

- не возвращает значение, но может иметь параметры и быть перегружаемым

Существуют два вида конструкторов - явные и неявные. Явно прописывая конструктор, Вы получаете возможность регулировать, **какие параметры и** **в каком количестве** нужно задать для создания объекта определенного класса.

1. *Использование super, final и this*

Имея дело с перегруженными конструкторами, удобно один конструктор вызывать из другого через ключевое слово **this**. При выполнении конструктора **this()** сначала выполняется перегруженный конструктор, который соответствует списку параметров. Затем выполняются операторы, находящиеся внутри исходного конструктора, если таковые существуют. Вызов конструктора **this()** должен быть первым оператором в конструкторе.

Eсли метод базового класса переопределен (имеет ту же сигнатуру) в производном классе, то такой метод базового класса можно вызвать из производного с помощью конструкции. При вызове super.метод() обращение производится к ближайшему суперклассу. «Super.id» – обращение к атрибуту.

При создании объектов производного класса, конструктор производного класса вызывает соответствующий конструктор базового класса с помощью ключевого слова Super(список параметров). Вызов конструктора базового класса из конструктора производного класса должен быть произведен в первой строке конструктора производного класса. Если конструктор производного класса явно не вызывает конструктор базового, то происходит вызов конструктора по умолчанию базового, то происходит вызов конструктора по умолчанию базового класса, в этом случае в базовом классе должен быть определен конструктор по умолчанию.

Нельзя создать подкласс для класса, объявленного со спецификатором **final**

1. *Java 8.*

Начиная с Java 8 кроме абстрактных методов мы также можем использовать в интерфейсах стандартные методы (default method метод- метод, тело которого представлено блоком – реализация по умолчанию для любого класса, который реализует интерфейс. Методы интерфейса по умолчанию отличны от конкретных методов, которые объявляются в классах. Статические методы (static methods) в интерфейсах – то же самое, что и static methods в абстрактном классе.

При реализации интерфейса, класс обязан реализовывать все методы интерфейса, иначе класс - абстрактный. Интерфейс может содержать внутренние классы и не абстрактные методы в них.

Абстрактный класс используют, когда нужна реализация по умолчанию. Интерфейс – когда классу необходимо нужно указать конкретное поведение.

1. *Что такое метод.*

Метод — это последовательность команд, которые вызываются по определенному имени. Можно сказать что это функция и процедура (в случае void метода).

Сигнатура метода в Java — это имя метода плюс параметры (причем порядок параметров имеет значение).  
В сигнатуру метода не входит возвращаемое значение, бросаемые им исключения, а также модификаторы.

Ключевые слова public, protected, private, abstract, static, final, synchronized, native, strictfp в т.ч. аннотации для метода — это модификаторы и не являются частью сигнатуры.

1. *Все о static*

Модификатор static в Java напрямую связан с классом, если поле статично, значит оно принадлежит классу, если метод статичный, аналогично - он принадлежит классу. Например, если поле count статично в классе Counter, значит, вы можете обратиться к переменной запросом вида: Counter.count. Конечно, следует учитывать модификаторы доступа.

Предположим, существует статический метод increment() в классе Counter, задачей которого является инкрементирование счётчика count. Для вызова данного метода можно использовать обращение вида Counter.increment(). Нет необходимости создавать экземпляр класса Counter для доступа к статическому полю или методу. Это фундаментальное отличие между статическими и НЕ статическими объектами (членами класса). Cтатические члены класса напрямую принадлежат классу, а не его экземпляру. То есть, значение статической переменной count будет одинаковое для всех объектов типа Counter.

1. *Основные моменты использования статических методов, полей и классов.* 
   1. Вы НЕ можете получить доступ к НЕ статическим членам класса, внутри статического контекста, как вариант, метода или блока. Результатом компиляции приведенного ниже кода будет ошибка. Это одна из наиболее распространённых ошибок допускаемых программистами Java, особенно новичками. Так как метод main статичный, а переменная count нет, в этом случае метод println, внутри метода main выбросит “Compile time error”.
   2. В отличие от локальных переменных, статические поля и методы НЕ потокобезопасны (Thread-safe) в Java. На практике это одна из наиболее частых причин возникновения проблем связанных с безопасностью мультипоточного программирования. Учитывая что каждый экземпляр класса имеет одну и ту же копию статической переменной, то такая переменная нуждается в защите - «залочивании» классом. Поэтому при использовании статических переменных, убедитесь, что они должным образом синхронизированы (synchronized), во избежание проблем, например таких как «состояние гонки» (race condition).
   3. Статические методы имеют преимущество в применении, т.к. отсутствует необходимость каждый раз создавать новый объект для доступа к таким методам. Статический метод можно вызвать, используя тип класса, в котором эти методы описаны. Именно поэтому, подобные методы как нельзя лучше подходят в качестве методов-фабрик (factory), и методов-утилит (utility). Класс java.lang.Math - замечательный пример, в котором почти все методы статичны, по этой же причине классы-утилиты в Java финализированы (final). **4**
   4. Другим важным моментом является то, что вы НЕ можете переопределять (Override) статические методы. Если вы объявите такой же метод в классе-наследнике (subclass), т.е. метод с таким же именем и сигнатурой, вы лишь «спрячете» метод суперкласса (superclass) вместо переопределения. Это явление известно как сокрытие методов (hiding methods). Это означает, что при обращении к статическому методу, который объявлен как в родительском, так и в дочернем классе, во время компиляции всегда будет вызван метод исходя из типа переменной. В отличие от переопределения, такие методы не будут выполнены во время работы программы.
2. *Циклы*

В Java существует три оператора цикла, которые очень важно хорошо знать для написания программ. **Цикл** — операторы выполняются последовательно: первый оператор функции выполняется первым, затем второй и так далее. Цикл используется в ситуации, когда Вам нужно выполнить блок кода несколько раз.

|  |  |
| --- | --- |
| **Цикл** | **Описание** |
| [while](http://proglang.su/java/loop-while) | Повторяет оператор или группу операторов, пока заданное условие является true. Цикл проверяет условие до выполнения тела цикла. |
| [for](http://proglang.su/java/loop-for) | Выполняет последовательность операторов несколько раз и сокращает код, которым управляет переменная цикла. |
| [улучшенный for](http://proglang.su/java/improved-loop-for) | Выполняет последовательность операторов несколько раз и сокращает код, которым управляет переменная цикла. |
| [do...while](http://proglang.su/java/loop-do-while) | Выполняется цикл while, за исключением того, что он проверяет условия в конце тела цикла. |

**Операторы цикла** — изменяют нормальное выполнение последовательности цикла в Java. Когда выполнение выходит из своей области, все объекты, которые были созданы автоматически в этой области будут уничтожены.

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| [break](http://proglang.su/java/operator-break) | Завершает работу цикла или оператора switch, и передаёт выполнение следующему, который находится сразу же после цикла или оператора switch. |
| [continue](http://proglang.su/java/operator-continue) | Заставляет цикл, пропустить оставшуюся часть его тела и сразу перепроверить вновь его состояние сначала. |

1. *Что такое поле/атрибут класса?*

**Поле класса** в ООП мире называется переменная, связанная с классом или объектом. Все данные объекта хранятся в его полях. Доступ к полям осуществляется по их имени. У полей есть альтернативные названия:

* атрибут класса
* переменная-член класса

Поля бывают собственные (обычные) и общие для всех объектов (экземпляров (instance’ов)) класса (статические).

1. *Что такое сигнатура метода*

Сигнатура метода — это имя метода плюс параметры (причем порядок параметров имеет значение). В сигнатуру метода не входит возвращаемое значение и модификатор static.

1. *Методы с переменным числом параметров.*

Возможность передачи в метод нефиксированного числа параметров позволяет отказаться от предварительного создания массива объектов для его последующей передачи в метод.

**void methodName(Тип[]… args){}**

В списке аргументов аргумент с переменным числом параметров должен быть самым последним

**void methodName(char s, int…args){}**

Методы с переменным числом аргументов могут быть перегружены.

1. *О чем говорит ключевое слово final?*

Модификатор 'final' обращается к классам, методам и переменным. Значение final изменяется от контекста к контексту, но основная идея та же самая: Особенности final не могут быть изменены.   
 Класс final. Класс, объявленный, объявленный с модификатором 'final' не может иметь подклассов. Таким образом модификатор 'final' добавляется к классу, когда его спецификация заморожена. Определение класса как final просто исключает наследование и ничего более. Однако, по причине предотвращения наследования, все методы в классе **final** неявно **final**, с этого момента нет способа отменить это.   
Метод final. При создании класса final не всё удовлетворяет нашим потребностям, мы действительно хотим только защитить некоторые из методов нашего класса от того, чтобы быть отменёнными, мы можем использовать ключевое слово final в объявлении метода, чтобы указать компилятору, что метод не может быть отменен подклассами. Метод final реализуется точно однажды, в объявлении класса. Такой метод не может быть отменен: подклассы не могут заменить метод новым определением. Любые частные методы в классе неявно final. Поскольку мы не можем обратиться к частному методу, мы не можем отменить его (компилятор дает сообщение об ошибках). Если класс final, то все его методы неявно final, то есть гарантируется неотменяемость в любом подклассе.   
Переменная final. Переменная final может быть установлена только однажды, разрешая нам объявить частые константы. Такую переменную можно оставить неназначенной в месте декларации, создавая *blank finals* (пустые переменные final). Но все переменные final должны быть назначены точно однажды. Эти переменные применяются главным образом в двух ситуациях: для предотвращения случайного изменения параметра метода, а также переменных, к которым обращаются анонимные классы.