1.

Ключевое слово this требуется для того, чтобы метод мог сослаться на вызвавший его объект.

Как правило, применять this нужно **в двух случаях:**

1. Когда у переменной экземпляра класса и переменной метода/конструктора одинаковые имена;
2. Когда нужно вызвать конструктор одного типа (например, конструктор по умолчанию или параметризированный) из другого. Это еще называется явным вызовом конструктора.

<https://javarush.ru/groups/posts/608-kljuchevoe-slovo-this-v-primerakh>

2.

extends is for *extending* a class.

implements is for *implementing* an interface

Difference: implements means you are using the elements of a **Java** Interface in your **class**. **Extends** means that you are creating a subclass of the base **class** you are **extending**. You can only **extend** one **class** in your child **class**, but you can implement as many interfaces as you would like.

Also java doesn't support **multiple inheritance** for classes. This is solved by using multiple interfaces.

public interface ExampleInterface {

public void doAction();

public String doThis(int number);

}

public class sub implements ExampleInterface {

public void doAction() {

//specify what must happen

}

public String doThis(int number) {

//specfiy what must happen

}

}

now extending a class

public class SuperClass {

public int getNb() {

//specify what must happen

return 1;

}

public int getNb2() {

//specify what must happen

return 2;

}

}

public class SubClass extends SuperClass {

//you can override the implementation

@Override

public int getNb2() {

return 3;

}

}

in this case

Subclass s = new SubClass();

s.getNb(); //returns 1

s.getNb2(); //returns 3

SuperClass sup = new SuperClass();

sup.getNb(); //returns 1

sup.getNb2(); //returns 2

I suggest you do some more research on **dynamic binding, polymorphism and in general inheritance in Object-oriented programming**

Static:

1. Если метод (переменная) имеет модификатор **static**, то это значит, что она объявляется **для класса**. Без этого модификатора - для объектов
2. 2. Из-за того, что статический метод/переменная связана с классом, а не объектом (см. пункт 1), их можно вызывать **без создания объекта класса**.э
3. А теперь, давайте продемонстрирует еще один принцип работы **static**- переменная с этим модификатором **одна для всех  объектов класса**.
4. Другим важным моментом является то, что вы НЕ можете переопределять (Override) статические методы.
5. <https://vertex-academy.com/tutorials/ru/modifikator-static-v-java/>
6. <https://javarush.ru/groups/posts/modifikator-static-java>

Private static final -> create this variable only once. private final -> create this variable for every object. First one saves memory, go for it.

FINAL

Суть **модификатора final** - сделать дальнейшее изменение объекта невозможным.  Если вы хотите, чтобы **после инициализации** никто **не мог бы изменить** вашу переменную, напишите слово "final"

#3

1. Библиотека **GSON** была разработана программистами Google и позволяет конвертировать объекты JSON в Java-объекты и наоборот.
2. The **javax**.**swing** package provides classes for **java swing** API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.
3. **Package java.awt Description**

Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components. See Component for a detailed description of properties that all AWT components share.

Some components fire events when a user interacts with the components. The AWTEvent class and its subclasses are used to represent the events that AWT components can fire. See AWTEvent for a description of the AWT event model.

A container is a component that can contain components and other containers. A con tainer can also have a layout manager that controls the visual placement of components in the container. The AWT package contains several layout manager classes and an interface for building your own layout manager. See Container and LayoutManager for more information.

Each Component object is limited in its maximum size and its location because the values are stored as an integer. Also, a platform may further restrict maximum size and location coordinates. The exact maximum values are dependent on the platform. There is no way to change these maximum values, either in Java code or in native code. These limitations also impose restrictions on component layout. If the bounds of a Component object exceed a platform limit, there is no way to properly arrange them within a Container object. The object's bounds are defined by any object's coordinate in combination with its size on a respective axis.

1. Interface **ActionListener**. ... The class that is interested in processing an action**event** implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action **event** occurs, that object's actionPerformed method is invoked.
2. public class **MouseEvent**

extends [InputEvent](https://docs.oracle.com/javase/7/docs/api/java/awt/event/InputEvent.html)

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. For lightweight components, such as Swing's components, mouse events are only dispatched to the component if the mouse event type has been enabled on the component. A mouse event type is enabled by adding the appropriate mouse-based EventListener to the component ([MouseListener](https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html" \o "interface in java.awt.event) or [MouseMotionListener](https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseMotionListener.html" \o "interface in java.awt.event)), or by invoking [Component.enableEvents(long)](https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html" \l "enableEvents(long)) with the appropriate mask parameter (AWTEvent.MOUSE\_EVENT\_MASK or AWTEvent.MOUSE\_MOTION\_EVENT\_MASK). If the mouse event type has not been enabled on the component, the corresponding mouse events are dispatched to the first ancestor that has enabled the mouse event type.

1. Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

Предположим, вы были:

1. Чтение сетевого файла и его отсоединение.
2. Чтение локального файла, который больше не доступен.
3. Использование некоторого потока для чтения данных и другого процесса закрывает поток.
4. Попытка чтения/записи файла, но не имеет разрешения.
5. Попытка записать файл, но дисковое пространство больше не доступно.
6. **java.util.ArrayList** - самый популярный вид списков. По названию пространства имён видно, что данный класс относится к Java.

**ArrayList** - автоматически расширяемый массив. Вы можете работать с массивом, но при этом не используются квадратные скобки.

<http://developer.alexanderklimov.ru/android/java/arraylist.php>

#4

public interface **WindowConstants**

Constants used to control the window-closing operation. The setDefaultCloseOperation and getDefaultCloseOperation methods provided by JFrame, JInternalFrame, and JDialog use these constants. For examples of setting the default window-closing operation, see [Responding to Window-Closing Events](https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html#windowevents), a section in The Java Tutorial.

# setDefaultCloseOperation()

The setDefaultCloseOperation() method is used to specify one of several options for the close button. Use one of the following constants to specify your choice:

* JFrame.EXIT\_ON\_CLOSE — Exit the application.
* JFrame.HIDE\_ON\_CLOSE — Hide the frame, but keep the application running.
* JFrame.DISPOSE\_ON\_CLOSE — Dispose of the frame object, but keep the application running.
* JFrame.DO\_NOTHING\_ON\_CLOSE — Ignore the click.

If you forget to call setDefaultCloseOperation() you will get JFrame.HIDE\_ON\_CLOSE by default. This can be frustrating, because it looks like you have "killed" the program, but it keeps on running, and you see no frame.

#5

setSize(int width, int height) — устанавливает размеры окна. Если не задать размеры, окно будет иметь нулевую высоту независимо от того, что в нем находится и пользователю после запуска придется растягивать окно вручную. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка.

#6

#### setLocationRelativeTo

public void setLocationRelativeTo([Component](https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html) c)

Sets the location of the window relative to the specified component according to the following scenarios.

The target screen mentioned below is a screen to which the window should be placed after the setLocationRelativeTo method is called.

* If the component is null, or the GraphicsConfiguration associated with this component is null, the window is placed in the center of the screen. The center point can be obtained with the [GraphicsEnvironment.getCenterPoint](https://docs.oracle.com/javase/7/docs/api/java/awt/GraphicsEnvironment.html" \l "getCenterPoint()) method.
* If the component is not null, but it is not currently showing, the window is placed in the center of the target screen defined by the GraphicsConfiguration associated with this component.
* If the component is not null and is shown on the screen, then the window is located in such a way that the center of the window coincides with the center of the component.

If the screens configuration does not allow the window to be moved from one screen to another, then the window is only placed at the location determined according to the above conditions and its GraphicsConfiguration is not changed.

**Note**: If the lower edge of the window is out of the screen, then the window is placed to the side of the Component that is closest to the center of the screen. So if the component is on the right part of the screen, the window is placed to its left, and vice versa.

If after the window location has been calculated, the upper, left, or right edge of the window is out of the screen, then the window is located in such a way that the upper, left, or right edge of the window coincides with the corresponding edge of the screen. If both left and right edges of the window are out of the screen, the window is placed at the left side of the screen. The similar placement will occur if both top and bottom edges are out of the screen. In that case, the window is placed at the top side of the screen.

The method changes the geometry-related data. Therefore, the native windowing system may ignore such requests, or it may modify the requested data, so that the Window object is placed and sized in a way that corresponds closely to the desktop settings.

**Parameters:**

c - the component in relation to which the window's location is determined

**Since:**

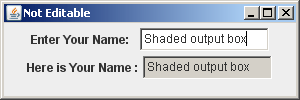
1.4

**See Also:**

[GraphicsEnvironment.getCenterPoint()](https://docs.oracle.com/javase/7/docs/api/java/awt/GraphicsEnvironment.html#getCenterPoint())

#7

# The setEditable() Method



Nothing happens because the lower text field does not have a registered listener. You could modify the program so it has a listener, but this would not follow the original design for the program. A better idea (for this program) is to prevent the user from entering text into the wrong text field. This can be done with the setEditable() method:

outText.setEditable( false );

Here outText is a variable that refers to a JTextField. The setEditable() method has one Boolean parameter. If the parameter is false, then the user cannot type into the field. However, the setText() method still works, so the program can use the field to display results. Here is an excerpt from the program:

public class Repeater extends JFrame implements ActionListener

{

JLabel inLabel = new JLabel( "Enter Your Name: " ) ;

JTextField inText = new JTextField( 15 );

JLabel outLabel = new JLabel( "Here is Your Name :" ) ;

JTextField outText = new JTextField( 15 );

public Repeater( String title) // constructor

{

super( title );

setLayout( new FlowLayout() );

add( inLabel ) ;

add( inText ) ;

add( outLabel ) ;

add( outText ) ;

inText.addActionListener( this );

setDefaultCloseOperation( JFrame.EXIT\_ON\_CLOSE );

}

.....

}

**Текстовые поля редактируются по умолчанию. Код setEditable (false) делает TextField неизменным. Он по-прежнему выбирается, и пользователь может копировать данные из него, но пользователь не может напрямую изменять содержимое TextField.**

**log.setLineWrap(true);**Cпособность переносить текст (то есть длинные строки будут укладываться в несколько строк вместо одной, уходящей за границы компонента. Этой способностью управляет метод setLineWrap(boolean lineWrap).

**fieldinput.addActionListener(this);**

The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's **addActionListener** method. When the action event occurs, that object's **actionPerformed** method is invoked.

**add()** method of Java.util.ArrayList class. This method is used for adding an element to the ArrayList.

**#8**

<https://javaswing.wordpress.com/2009/11/08/borderlayout/>

**#9**

<https://javaswing.wordpress.com/2009/08/04/jscrollpane_using/>

**#10**

Флаг focusable указывает, может ли компонент получить фокус, если это требуется для этого. Компонент JPanel по умолчанию настраивается, поэтому при его установке на true ничего не изменится. Компонент, который не фокусируется, не может получить фокус.

**Пример**

Скажем, вы внедрили диалог с несколькими текстовыми полями и хотите, чтобы пользователь вводил текст. Когда пользователь начинает вводить текст, одно текстовое поле должно иметь фокус приложения: это будет поле, которое получает ввод на клавиатуре.

**#11**

**Gson() -** библиотека Java с открытым исходным кодом для сериализации и десериализации объектов Java в JSON.

Gson - это небольшая java библиотека, которая позволяет конвертировать java объекты в их JSON представление, равно как и создавать объекты на основании их json представления.

Изначально Gson был разработан в Google и использовался в нескольких внтуренних проектах. Через некоторое время было принято решение отдать библиотеку в open-source, чтобы она и дальше развивалась.

Основным классом библиотеки есть одноименный класс Gson. Для того, чтобы создать экземпляр класса нужно воспользоваться одним из двух способов:

* Gson gson = new Gson();
* Gson gson = new GsonBuilder().create();

Первый способ создаст инстанс класса с настройками по умолчанию, а второй способ позволит применить некоторые настройки. О настройках расскажу чуть ниже.

Основные методы, которые используются для сериализации и десериализации java-объектов, называются toJson и fromJson.

<http://www.javenue.info/post/gson-json-api>

**#12**

Please advice how to convert a String to JsonObject using gson library.

You don't need to use JsonObject. You should be using Gson to convert to/from JSON strings and your own Java objects.

**#13**

**printStackTrace**() The **printStackTrace**() method of **Java**.lang.Throwable class used to print this Throwable along with other details like class name and line number where the exception occurred means its backtrace. This method prints a stack trace for this Throwable object on the standard error output stream. Используя printStackTrace(), вы получаете всю стекцию исключения, что очень полезно для отладки.

**#14**

<http://developer.alexanderklimov.ru/android/java/thread.php>

The reload() method is used to reload the current document.

The reload() method does the same as the reload button in your browser.

By default, the reload() method reloads the page from the cache, but you can force it to reload the page from the server by setting the forceGet parameter to **true**: location.reload(true).

When the thread is required to read from main memory is defined in terms of the synchronization order and happens before order. Basically it says that in order for a read to yield the value that was last written, the write needs to happen-before the read.

The happens-before relation is roughly speaking defined in terms of locking/unlocking actions and (with a few exceptions) boils down to the usage of synchronized methods and blocks. Unless you're dealing with volatile variables, the bottom line is usually that you need to synchronize all access to shared data, preferably through AtomicBoolean, a BlockingQueue or some other java.util.concurrent class.

**#15**

В Java оператор "==" возвращает значение типа boolean - результат сравнения ссылок на объекты (кроме примитивов), т.е. данный оператор **не сравнивает на равенство внутреннее содержимое объектов**, а просто проверяет указывают ли ссылки на один и тот-же объект. Таким образом, при сравнении объектов в Java оператор "==" вернет true лишь в том случае, когда ссылки указывают на один и тот-же объект.

<https://ru.stackoverflow.com/questions/496080/%D0%9A%D0%B0%D0%BA-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80>

**#16**

The **removeAllElements()** method is used to remove all components from this vector and sets its size to zero.This method is identical in functionality to the clear method.

**#17**

**Synchronized**(с англ. "синхронизированный") - это ключевое слово, которое позволяет заблокировать доступ к методу или части кода, если его уже использует другой поток.

<https://vertex-academy.com/tutorials/ru/klyuchevoe-slovo-synchronized-java/>