toString() returns the String representation of the object that calls it. When want to know what the elements in an array are, I should iterate the array for printing each element.

<https://stackoverflow.com/questions/7520432/what-is-the-difference-between-and-equals-in-java>

<https://www.geeksforgeeks.org/difference-equals-method-java/>

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html>

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

<https://www.geeksforgeeks.org/difference-equals-method-java/>

var str = "Java 10"; // infers String

var list = new ArrayList<String>(); // infers ArrayList<String>

var stream = list.stream(); // infers Stream<String>s

<https://dzone.com/articles/finally-java-10-has-var-to-declare-local-variables>

<https://stackoverflow.com/questions/35706201/is-there-any-package-to-store-the-set-of-java-keywords>

<https://intellij-support.jetbrains.com/hc/en-us/articles/206544879-Selecting-the-JDK-version-the-IDE-will-run-under>

switch boot jdk; boot

do-while loop and for loop

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#get-int->

<https://docs.oracle.com/javase/8/docs/api/java/sql/Array.html>

<https://www.tutorialspoint.com/java/java_loop_control.htm>

some package will import automatically when I use a method in the package, but some will not; why? (to clarify)

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>

three ints() methods in Random

source code of Random class:

<http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/Random.java>

source code of String class:

<http://hg.openjdk.java.net/jdk7u/jdk7u6/jdk/file/8c2c5d63a17e/src/share/classes/java/lang/String.java>

<https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps>

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html#toArray-->

<https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html>

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

Threads can be created by using two mechanisms:

1. Extending the Thread class

2. Implementing the Runnable Interface

We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

1. If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.

2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.

It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class. Important terminology: Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).

<https://www.geeksforgeeks.org/multithreading-in-java/>

<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>

implement Callable Interface, via FutureTask () to create Thread

use ExecutorService, Callable, and Future to implement creating Thread with return values

<https://www.quora.com/How-do-I-write-a-java-test-class>

<https://stackoverflow.com/search?q=java+Test+a+b+c>

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html>

Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).

The body of a switch statement is known as a switch block. A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label.

Another point of interest is the break statement. Each break statement terminates the enclosing switch statement. Control flow continues with the first statement following the switch block. The break statements are necessary because without them, statements in switch blocks fall through: All statements after the matching case label are executed in sequence, regardless of the expression of subsequent case labels, until a break statement is encountered. The program SwitchDemoFallThrough shows statements in a switch block that fall through. The program displays the month corresponding to the integer month and the months that follow in the year:

Technically, the final break is not required because flow falls out of the switch statement. Using a break is recommended so that modifying the code is easier and less error prone. The default section handles all values that are not explicitly handled by one of the case sections.

<https://blog.csdn.net/u014401141/article/details/72819746>

<https://blog.csdn.net/wanght89/article/details/79035332>

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/catch-multiple.html>

先小异常再大异常，catch捕捉有个规则，当异常对象是catch后面的异常的实例的话，其他的catch块不会再执行

<https://www.tutorialspoint.com/java/java_basic_operators.htm>

<https://www.geeksforgeeks.org/constructors-in-java/>

<https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/Exception.html>

<https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/>

Exception occurred in try-block is not handled in catch block: In this case, default handling mechanism is followed. If finally block is present, it will be executed followed by default handling mechanism.

<https://www.geeksforgeeks.org/compare-two-strings-in-java/>

Using String.equals() :In Java, string equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are same then it returns true. If all characters do not match, then it returns false.

Using String.equalsIgnoreCase() : The String.equalsIgnoreCase() method compares two strings irrespective of the case (lower or upper) of the string. This method returns true if the argument is not null and the contents of both the Strings are same ignoring case, else false.

Using Objects.equals() : Object.equals(Object a, Object b) method returns true if the arguments are equal to each other and false otherwise. Consequently, if both arguments are null, true is returned and if exactly one argument is null, false is returned. Otherwise, equality is determined by using the equals() method of the first argument.

Using String.compareTo() :

Main difference between .equals() method and == operator is that one is method and other is operator.

One can use == operators for reference comparison (address comparison) and .equals() method for content comparison.

In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

<https://stackoverflow.com/questions/767372/string-equals-versus>

<https://www.geeksforgeeks.org/bitwise-shift-operators-in-java/>

1) >> (Signed right shift) In Java, the operator ‘>>’ is signed right shift operator. All integers are signed in Java, and it is fine to use >> for negative numbers. The operator ‘>>’ uses the sign bit (left most bit) to fill the trailing positions after shift. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler. For example, if binary representation of number is 10….100, then right shifting it by 2 using >> will make it 11…….1.

2) >>> (Unsigned right shift) In Java, the operator ‘>>>’ is unsigned right shift operator. It always fills 0 irrespective of the sign of the number.

<http://www.runoob.com/w3cnote/java-init-object-process.html>

<https://segmentfault.com/a/1190000009888357>

<https://docs.oracle.com/javase/7/docs/api/java/lang/Character.html>

<https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html>

A thread-safe, mutable sequence of characters. A string buffer is like a String, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

Whenever an operation occurs involving a source sequence (such as appending or inserting from a source sequence) this class synchronizes only on the string buffer performing the operation, not on the source.

Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger. As of release JDK 5, this class has been supplemented with an equivalent class designed for use by a single thread, StringBuilder. The StringBuilder class should generally be used in preference to this one, as it supports all of the same operations but it is faster, as it performs no synchronization.

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

A String represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs (see the section Unicode Character Representations in the Character class for more information). Index values refer to char code units, so a supplementary character uses two positions in a String.

<https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html>

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

<https://blog.csdn.net/lzm1340458776/article/details/26276581>

<https://stackoverflow.com/questions/7569459/can-i-have-multiple-run-methods-in-a-class>

<https://blog.csdn.net/zzq900503/article/details/40377967>

<https://blog.csdn.net/J080624/article/details/81836930>

(8) 表达式1,表达式2,表达式3的类型可以不同，进行自动类型转换

此时条件表达式的值的类型为它们中较高的类型，什么意思？

条件表达式最终产生的类型取决于下述情况：

if 第二个操作数和第三个操作数有相同的类型（可以都为null），那么它就是条件表达式的类型。

else if 两个操作数中有一个的类型为原始类型T，而另一个为T的装箱类型，那么条件表达式的类型就是T。

else if 其中一个操作数是编译时null类型，另一个为引用类型，那么条件表达式的类型就是该引用类型。

else if 两个操作数都可转化为数字类型，那么分为以下情况：

if 其中一个类型为byte 或 Byte，另一个类型为short 或 Short，那么条件表达式类型为short。

else if 其中一个类型为T，T为byte、short 或 char，另一个类型为int类型的常量表达式，且可用T来表达（在T可表示的范围内），那么条件表达式类型为T。

else if 其中一个类型为T，T为Byte、Short 或 Character，另一个类型为int类型的常量表达式，且可用U来表达（U为T的拆箱类型），那么条件表达式类型为U。

else 对两个操作数使用二元数值提升机制（并没有真的去转换类型），得到的相同数值类型就是条件表达式的类型。

否则，双目数值提升（binary numeric promotion）会被用于表达式的类型中，条件表达式的类型是第二个和第三个提升后的类型。

注意：双目数值提升时进行拆箱转换和值集转换（value set conversion）。

以上操作仅用于编译器判断条件表达式的最终类型T，只有在最终选择的操作数（第二个表达式的或第三个表达式的）与T不符时才会进行自动拆箱/类型提升操作。

数值计算表达式的数据类型自动提升，需要注意下面规则：

所有的 byte, short, char 型的值将被提升为 int 型；

整数操作，如果有一个操作数是 long 型， 计算结果是 long 型；但是要注意，如果另一个数是float 或者 double 类型时，则计算结果转化为 float 或者 double 类型，此时与 long 类型不兼容，不能直接复制给 long 类型变量。

如果有一个操作数是 float 类型，计算结果是 float 类型，前提是另一个数不能为 double 类型，否则计算结果则为 double 类型。float 类型可以自动转化为 double，double 类型不能自动转换为 float 类型。

如果有一个操作数是 double 类型，则计算结果就为 double 类型。

总结，高精度的数据类型无法自动转换为低精度的数据类型；而低精度的数据类型可以自动转换为高精度的数据类型。

运算中的类型提升通常都是将低于int位数的类型提升为int，高于int的拆箱后保持不变，两边操作数位数不同则升为高精度的那一个类型。

<https://blog.csdn.net/zhangqilugrubby/article/details/80500505>

（看解析，学习线程和并发，再返回理解解析）

BCD (character encoding) BCD ("Binary-Coded Decimal"), also called alphanumeric BCD, alphameric BCD, BCD Interchange Code, or BCDIC, is a family of representations of numerals, uppercase Latin letters, and some special and control characters as six-bit character codes.

<https://en.wikipedia.org/wiki/BCD_(character_encoding)>

ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices.

<https://en.wikipedia.org/wiki/ASCII>

GBK is an extension of the GB2312 character set for simplified Chinese characters, used in the People's Republic of China. It includes all unified CJK characters found in GB13000.1-93, i.e. ISO/IEC 10646:1993, or Unicode 1.1.

<https://en.wikipedia.org/wiki/GBK_(character_encoding)>

Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.

<https://en.wikipedia.org/wiki/Unicode>

<https://docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html>

字符编码英文全称为Chacter encoding，设定一个默认的字符集，用字符集的每一个字符来指定集合中的某一个对象，来便于文本在计算机中的存储和网络通信。

BCD: 这是一种常用于会计系统中的编码方式，与java关系不大，错误

ASCII码1961年提出，是目前使用最广泛的西文字符集，但不是java使用的编码

GBK为一种汉子编码格式，包含了大量的汉子编码，java也并未使用它最为编码格式Unicode编码是一种将世界上所有符号都纳入其中的编码格式来避免编码不匹配的乱码问题，java选用的就是这种编码格式，但存储和传输所占用的资源就相对较大

类、方法修饰符——abstract、final

<https://www.quora.com/Can-we-declare-abstract-class-as-final-If-no-why>

A final class is considered complete and can not be subclassed (It's methods can not be overridden ). In case of abstract class, we have to proved implementation to abstract methods in subclasses. NOTE :- A final class can not have abstract methods and an abstract class can not be declared final.

interface——接口定义

<https://www.geeksforgeeks.org/interfaces-in-java/>

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

Interfaces specify what a class must do and not how. It is the blueprint of the class.

An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.

If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.

A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

To declare an interface, use interface keyword. It is used to provide total abstraction. That means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface. To implement interface use implements keyword.

Why do we use interface ?

It is used to achieve total abstraction.

Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .

It is also used to achieve loose coupling.

Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

Prior to JDK 8, interface could not define implementation. We can now add default implementation for interface methods. This default implementation has special use and does not affect the intention behind interfaces.

Suppose we need to add a new function in an existing interface. Obviously the old code will not work as the classes have not implemented those new functions. So with the help of default implementation, we will give a default body for the newly added functions. Then the old codes will still work.

Another feature that was added in JDK 8 is that we can now define static methods in interfaces which can be called independently without an object. Note: these methods are not inherited.

Important points about interface or summary of article:

We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.

A class can implement more than one interface.

An interface can extends another interface or interfaces (more than one interface) .

A class that implements interface must implements all the methods in interface.

All the methods are public and abstract. And all the fields are public, static, and final.

It is used to achieve multiple inheritance.

It is used to achieve loose coupling.

New features added in interfaces in JDK 9

From Java 9 onwards, interfaces can contain following also

Static methods

Private methods

Private Static methods

<https://www.geeksforgeeks.org/referencing-subclass-objects-subclass-vs-superclass-reference/>

First approach (Referencing using Superclass reference): A reference variable of a superclass can be used to a refer any subclass object derived from that superclass. If the methods are present in SuperClass, but overridden by SubClass, it will be the overridden method that will be executed.

Second approach (Referencing using subclass reference) : A subclass reference can be used to refer its object.