Java Note

Every line of code that can actually run needs to be inside a class.

A public class must be declared inside a file with the same name, or an error will occur.

If write only one file, names are not necessarily the same.

Standard start:

public class Main {

public static void main(String[] args) {

System.out.println(“Goodbye, World!”);

}

}

public means anyone can access it.

Static means you can run this method without creating an instance of Main, because this method belongs to a class, not its instances.

System is a pre-defined class

Out is a static variable within System

Println is a method of out

Primitives:

Number (signed integer): byte(1byte), short(2bytes), int(4bytes), long(8bytes)

Float: float(4bytes), double(8bytes)

Character: char(2bytes Unicode) ‘\u0000’-‘\uffff’ (usage of Unicode char), unsigned

True or false: Boolean(1byte)

Default float type: double. So use 1.0f or (float)1.0 for float type.

Float f = 1.0;//Will not work

The same applies to long type

String type: String s1 = new String(“”);

No operator overloading in Java! Only primitives and Strings like: ”I am ” + 5 + “ years old”.

Boolean is a single type:

int children = 0;

Boolean b = children; // Will not work

if (children) { // Will not work

{}can be omitted in one-line statements including if else, while and …, just like in C++.

?:still exists

when using objects: == judges whether they are the same object, objA.equal(objB) judges whether the contents are the same.

Arrays in Java are also objects:

Int[] arr = new int[10];

Arr.length is always available.

Int[]arr = {1,2,3,4,5}; //also work

All three expression in for loop are optional.

Do{}while(condition);

Int[] arr = {1,2};

For (int el : arr) {}//also work when iterating on elements inside an array

All function definitions must be inside classes.

For primitives, assignment statements mean copy, but for objects, they mean share reference on the same object. Parameters acts exactly like assignment statements at the beginning of methods.

Non-static: can only be called by objects, can access fields of an object.

Create an instance: new

All classes that don't explicitly define a constructor has a default constructor that does nothing. If a constructor is defined, no default constructor will be created. Constructor: no return value, same name as class, can have multiple constructors.

This: generally the same, plus, we can use it as the first line within a constructor to call a different constructor

Point() {

this(0, 0);

}

private means only the class itself can access the variable or method.

Default: public

Commands:

Javac nameOfFile.java //compile and create nameOfFile.class containing java code

Java nameOfClass arg0 arg1 arg2//run code. nameOfFile = name of Class

A subclass does not inherit the private members of its parent class.

Extends: signal inherit

Dynamic binding:

shape a = new square();

a.area calls square’s area method instead of shape’s area method.

Try{}

Catch(nameOfTypeOfException nameOfExceptionInstanceForReference){}//can have many

finally{}//will always execute

To throw an exception: throw new xxxException(“”);

JRE = JVM + Required Library to run Application.

JDK = JRE + Required Library to develop Java Application.

Jvm is machine specific, it generate machine specific code from machine neutral bytecode generated by compiler, namely .class files.

<Access Modifier> class <Class\_Name> extends

                 <Super\_Class\_Name> implements <Interface\_Name>

one public class for each file (same name if contains a pubic class) but multiple non-public classes

begin a source file: package, import, classes. package and import are effective to all classes in a file

A public class can be seen by all *classes from all packages*

A class with default access can be seen only by classes within the same package.

An Abstract Class can not be instantiated.

A Final Class *can not be subclassed*.

A Class can not be both *final and abstract*.

Constructors cannot have Non Access Modifiers while Methods can.

Java naming convention, Method names should be camelcase while Constructor names should start with capital letter.

*A Method can have the same name as the Class name.*

Super() and this() statements are just the same, must be the first line, obey parameter rules, can’t occur together. This() links constructors in the same class while super links to its super class. If a constructor has neither a this() nor a super() construct as its first statement, then a super() call to the default constructor in the superclass is inserted. If a class only defines non-default constructors, its subclasses must then explicitly call a Superclass constructor, using the super() construct with the right arguments.

The only action taken by the implicit default Constructor is to call the Superclass Constructor using the super() call.

Variable definition:

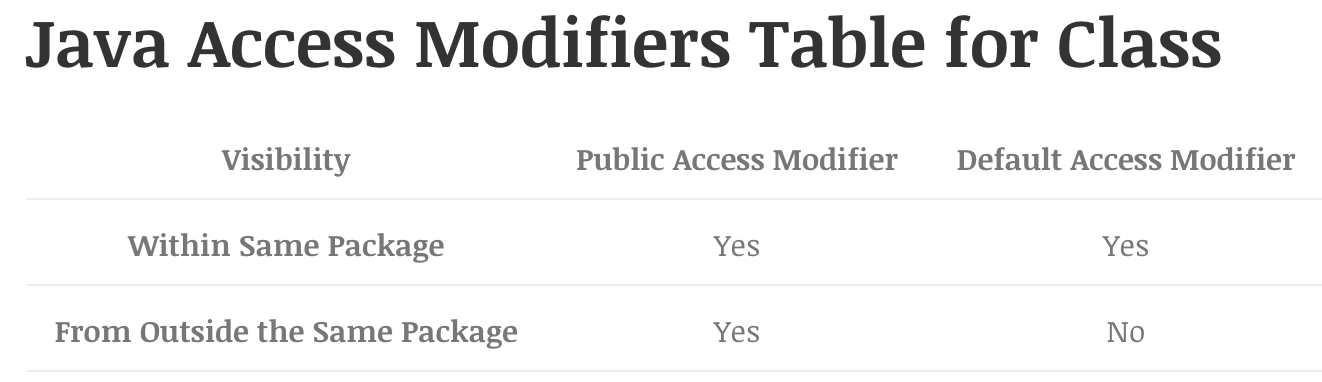
Access modifier, non-access modifier, type, name

Final is the Only modifier that can be applied to a local variable (inside a method).

Access modifiers:

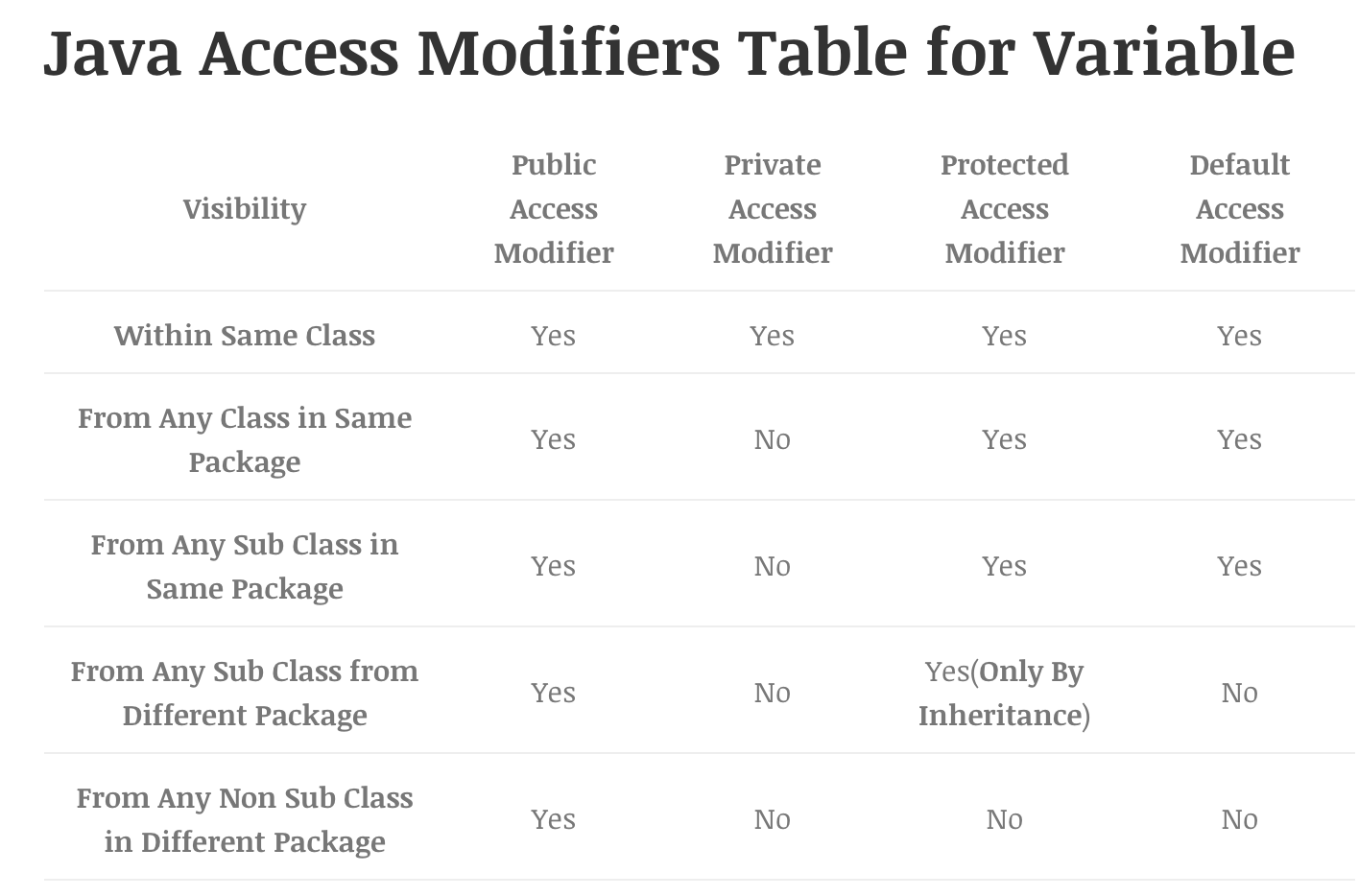
Public, protected, private, default.

Default is an access control which will be set when one does not specify any access modifier.

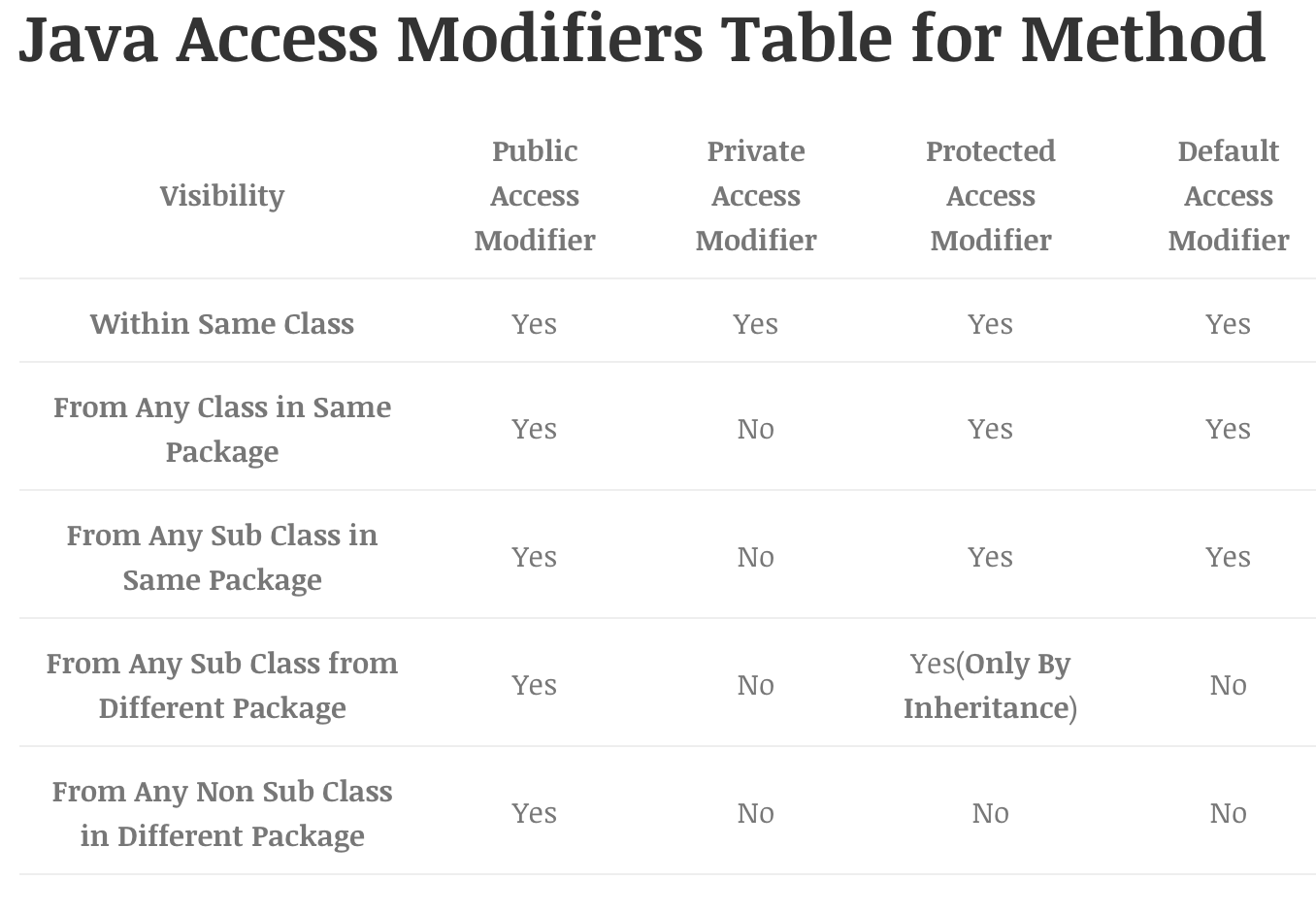


classes and interfaces can’t be private or protected (of course). methods and variables in interfaces can not be private or protected.

Visibility of the Class should be checked before checking the visibility of the variable defined inside that Class.



if not mentioned, yes means through inheritance (can be accessed by a.x by subclasses) or direct access (can be accessed by a.x by anyone).



non-access modifiers except static and transient:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Class | Method | Instance or local Variable or method  arguments |
| Final | cannot be extended | cannot be overridden by any subclass | Can’t change value |
| Abstract | Can be applied directly to classes. A class containing abstract methods become abstract class automatically. | No body, only signature. | x |
| Strictfp | forces floating point or floating point operation to adhere to IEEE 754 standard. | forces floating point or floating point operation to adhere to IEEE 754 standard. | x |
| Native | x | indicate that the method is implemented on a platform dependent code. | x |
| Synchronized | x | can be accessed by only one thread at a time. | x |

Method parameter or local variable will hide instance variable. In this case, use “this”.

Static: belongs to this class rather than instances. Can be accessed by instance.staticElement. Can’t be overridden. Can be redefined. If a Class contains any static blocks then that block will be executed only when the Class is loaded in JVM, creating multiple instances does not execute the static block multiple time. If Class.forName(“class\_name“) is called then the static block of the Class will get executed. Can be applied to: Method, Variable, Class nested within another Class, Initialization Block

Packages are used to avoid naming conflicts, to control access (Access Modifier) and to bundle groups of related types.

Another way to create an instance:

Class cls = Class.forName("JBTClass");

JBTClass obj = (JBTClass) cls.newInstance();

Class.forName actually loads the Class in Java but doesn’t create any Object. If you want to create the Object in this way Class needs to have a public default constructor.

Using clone:

need to implement Cloneable Interface, otherwise it will throw CloneNotSupportedException.

To implement:

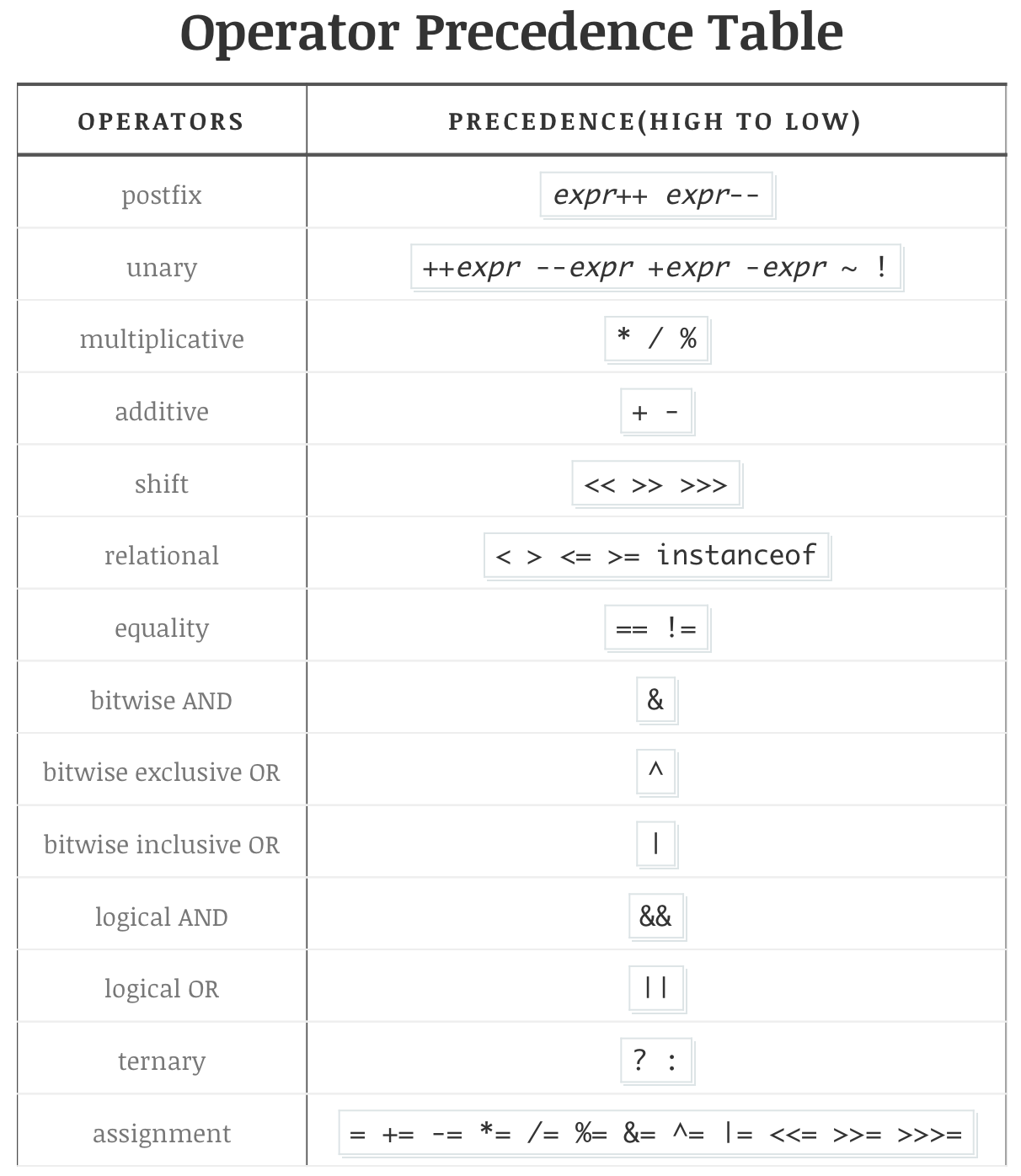
protected Object clone()throws CloneNotSupportedException{}

to use:

BTClassClone obj2 = (JBTClassClone) obj1.clone();

Using classloader:

obj = (JBTClassLoader) new CreateObjectWithClassLoader().getClass().getClassLoader().loadClass("JBTClassLoader").newInstance();



if()-else if()-else

interface: a contract.

<access> interface interfaceName extends OneOrMoreInterfaces{ //by default, interfaces are public

public static final var; //they must be so. If not declared, they are implicitly so.

<access> <static/default> void method(){…}; //can be implemented

<access> abstract void method(); //no implementation

void method(); //method default: public abstract

}

classes can extend only one class while they can implement more than one interface

Class implementing an interface can also be an abstract class.

An abstract class which is implementing an interface need not implement all abstract method.

A non abstract class which is implementing an Interface needs to follow some rules:

1.This class needs to provide concrete implementation of all abstract method.

2.All rules of Overriding needs to be followed. like access modifiers or exceptions, see below.

The java.lang.Object Class is always at the top of any Class inheritance hierarchy.

constructors and initializer blocks are not members of a Class, they are not inherited by a subclass.

The super keyword also references the current Object, but as an instance of the current class’s super class.

All default variables will be inherited by all subclasses in the same package only. Subclasses outside the package will not inherit any default member.

Overload means same name different arguments, overriding means covered by subclasses.

Overloaded methods:

Must change the argument list

Can change the return type, access modifier, can declare different exception

By definition, polymorphism applies to overriding not Overloading

Determining which overloaded Method will be invoked is decided at compile time on the basis of the reference type.

A method can be overloaded in Class or in SubClass.

In case of concrete subclass it is forced to implement all methods defined in abstract class if no other super class implemented it in hierarchy. Overriding sometimes referred as Run time Binding. Object type determines which overridden method will be invoked and that will be decided at the runtime.

Overriding method can’t have more restrictive access modifier than the method being overridden, but it can be less.

The argument list must exactly match that of the overridden method

Return type must be the same as, or subtype of the return type declared in overridden method in Super class.

Overriding method can not throw checked exception which is broader or new than those declared by the overridden method, but it can throw fewer or narrow checked exception.

Overriden method can throw any unchecked exception.

重写的方法能够抛出任何非强制异常 。但是，重写的方法不能抛出新的强制性异常，或者比被重写方法声明的更广泛的强制性异常，反之则可以。

Static Methods or variables do not take part in inheritance. So there’s no such thing as “override”, but you can surely re-invent it, like it never existed anywhere.

Want a parent class version? Use super.

Class’s innerclass:

Inner class instance has access to all member of the outer class

Inner class can be accessed only through live instance of outer class.

OuterClass outerclass = new OuterClass();

OuterClass.InnerClass innerclass = outerclass.new InnerClass();

This in inner class means innerclass’s this. Use outerclass.this to access outerclass’s elements.

Normal inner class will be treated like member of the outer class so it can have several Modifiers as opposed to Class: final abstract public private protected strictfp. Don’t get confused with the modifiers of Class and Inner Class. They are completely different.

Method’s innerclass:

Instances can only be created after full definition of this class.

can be instantiated within the method where it is defined and no where else.

Can’t use the variable defined in method where it is defined, except final variables.

available modifiers: final, abstract

inner class can be anonymous

Static nested class can not access non static member of outer class.

string:

All string literals in Java programs are implemented as instances of this class.

Strings are like constants.

String Buffer and String Builder can be used in place of String if lot of String Operation is to be performed.

important method: intern, length, toString, trim

Strings are objects, String Pool/ String Literal Pool are a collection of references to String objects because it’s good to “share” the same String object with multiple references

two ways to create: string literal, new. both points to string object on heap. the difference is when creating with literal, it first searches the string pool and see if this string already exists. if not, an additional reference will be cached in string pool for later search. So the rule is: if the string occurs often, use literal, otherwise use new. since string literals always have a reference in pool, they are not eligible for garbage collection.

‘character’, “string”

Where possible, it is recommended that String Builder should be used in preference to StringBuffer as it will be faster in most of the cases.

‘StringBuffer’ are safe for use by multiple threads(Thread Safe). The methods are synchronized. String Builder is not thread safe.

important methods for stringbuffer: append, insert, length

Array will always be an Object in Heap. No matter what it store, primitives or Object.

You cannot include size of array in declaration.

once created size of the array can not be changed, hence size of the array needs to be provided at the time of constructing it

arr = new int[10][];

arr[0][0] = 0;//work

Only first part(First Dimension is required) needs the size and not the all.

enum:can be used to replace constants or Boolean

public enum Workday {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY

}

use:

if(currentSchedule.getWorkday() == Workday.FRIDAY) {

literating:

for(Workday w : Workday.values()) {

System.out.println(w.name());

variables and methods in enum:

public enum Workday {

MONDAY("Monday"),

private final String representation; // has to be initialized during construction of the enum

private Workday(String representation) {

   this.representation = representation;

}

public String getRepresentation() {

   return this.representation;

}

}

You can have multiple properties / fields in an enum

enum can implement interfaces

Serialization and Deserialization: implement interface (actually nothing needs to be done because it doesn’t require any).

usage:

implements Serializable

FileOutputStream fileOut = new FileOutputStream("./employee.txt");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(emp);

out.close();

fileOut.close();

FileInputStream fileIn = new FileInputStream("./employee.txt");

ObjectInputStream in = new ObjectInputStream(fileIn);

emp = (Employee) in.readObject();

in.close();

fileIn.close();

If Super class implements Serializable then sub class are also Serializable automatically.

If Super class is not serializable then when sub class is de serialized then super class’s default constructor will be invoked. Hence all variable will get default value and reference will be null.

serialVersionUID is a number associated to each serializable class. This number is used during deserialization process to verify that the sender and receiver of a serialized object have loaded class for that object which is compatible with respect to serialization.

Defining a serialVersionUID field in serializable class is optional. if it is defined explicitly, it should be long static final. If there is no serialVersionUID field defined explicitly then serialization runtime will calculate default value for that class, which can vary based on compiler implementation. Hence it is advisable to define serialVersionUID. It is advised to use private access modifier. Array classes cannot declare an explicit serialVersionUID so the requirement for matching serialVersionUID values is waived for array classes. If there is a difference between serialVersionUID of loaded reciever class and corresponding sender class then InvalidClassException will be thrown. If you want to change the class a little bit while still want it to work, add an explicit serialVersionUID. ifferent class can have same serialVersionUID.

Transient fields state will not be saved while serialization process and default value will be assigned to same variable while de-serialization.

Static variables value can be stored while serializing if it is provided while initialization.

If variable is defined as Static and Transient both, then static modifier will govern the behavior. final instance variables will be saved regardless of transient, even final variables in interface will be saved.

Thread can be of two types. User Thread, Daemon Thread

JVM exit an application only when all User Thread are complete. JVM doesn’t care about the status of Daemon Thread.

A thread can be defined in two ways: 1Extend the java.lang.Thread class & Override the run() method, the problem with this approach is that class can not extend any more class. 2.Implement the Runnable interface & Override the run() method.

run method can be overloaded in class. but only run() method(without argument) will be consider by JVM.

To create a thread instance (will not start): 1.Extend Thread class: Thread t = new myThread(). 2.Implement Runnable interface: myRunnable r = new myRunnable(); Thread t = new Thread(r); either way, you get an instance of thread class. each of this corresponding to a thread.

Single runnable instance can be passed to multiple Thread object. The Thread class itself implements Runnable. This means that you could pass a Thread to another Thread’s constructor.

To start a thread: call start(). call run()? won’t work.

Thread class methods:

public static void sleep(long millis) throws InterruptedException //current running Thread will sleep

\* public static void yield() //stop the currently running thread to allow other threads of the same priority to run. The scheduler will make sure that Thread with the highest priority(1-10) runs first.

\* public final void join() throws InterruptedException //let one thread “join onto the end” of another thread. t.join() means currently running thread joins to the end of t

\* public final void setPriority(int newPriority)

Object class methods:

\* public final void wait() throws InterruptedException

\* public final void notify()

\* public final void notifyAll()

There is only one lock per object: one for class object and one for each instance of this class. Once a thread got the lock of an object no other thread can enter the synchronise block/method of given object. Only methods/block can be synchronized. Thread can access non synchronised block even if one Thread got the lock of given object. If a thread goes to sleep, it holds any locks it has. A thread can have Lock of different object at the same time. Threads calling non-static synchronized methods in the same class will only block each other if they’re invoked using the same instance. As there is only one lock per instance. Threads calling static synchronized methods in the same class will always block each other— As there is only one lock per class. join(), sleep(), yield() these methods keep locks, while wait release Lock. start() method can be called on Thread Object only once.

every class loaded in Java has a corresponding instance of java.lang.Class representing that class.

Object-Throwable-Exception-differentException:

-Error

Checked Exception

Checked exceptions are subclass’s of Exception excluding RuntimeException and its subclasses.

Checked Exceptions force programmers to deal with.

When a checked exception occurs, the method must either catch the exception and take the appropriate action, or pass the exception on to its caller.

Unchecked Exception

Unchecked exceptions are Error and RuntimeException and any of its subclasses.

Compiler doesn’t force the programmers to either catch or declare it.

Programmers may not even know that the exception could be thrown.

exception propagation: call stack

Catch block should be order in the form of most specific to most general.

List : Ordered, Duplicates are allowed, Indexed

Sets : May or may not Ordered. Duplicates are not allowed.

Maps : Duplicate keys are not allowed.

Queue : Ordered by FIFO or priority.

ArrayList : Fast Iteration & Fast Random Access.

Vector: Synchronized Method.

LinkedList : Good for implementing Stack and Queue.

HashSet : Fast Access, No Duplicates, No Ordering.

LinkedHashSet : No Duplicates, Iterates by insertion order.

TreeSet : No Duplicates, Iterates in sorted order.

collection and set are both interfaces.

switch: integer types or char or enum.

What is package?

<http://javabeginnerstutorial.com/core-java-tutorial/java-class-object-tutorial/>

what are <static initilizar block> and <ananymous\_block>?

runtimebinding can only apply to abstract methods in c++ but not in java

first character of the name of a class should be capitialized

first character of the name of a method should not be capitialized

variables can have same name but different case, but classes can not

enum can be declared in class

instanceof: test whether an object is an instance of a class

throw != throws: exceptions that can be thrown by a method

package: classify the classes defined in this file

import path include package name and class name, \*means any.

can use Byte.SIZE or Byte.MIN\_VALUE. int is Integer, char is Character.

name of constants should be capitalized

011=9

0x11=17

local variables do not have default values. They are on stack. other variables have intrinsic default values.

static variables are stored in static memory area, created at the beginning of program and destroyed when program ends. value can be designated when declare or construct or in static block.

by default, variables in interfaces are public static final, and methods in interfaces are public.

to use public classes in different packages, import them.

method access modifiers in subclasses must be looser or equal. private method will not be inherited.

final: final references can not be changed but variables inside that referenced object can be changed.

abstract methods can not have body, classes containing an abstract method must be abstract, abstract classes can have non-abstract methods, classes can not be both abstract and final, abstract methods can not be final. non-abstract subclasses of an abstract class must implement all its abstract methods.

volatile: r&w always up to date. so

volatile Boolean active = true;

while (active) {}

can be stopped by active = false of another thread.

bit operators can be applied to char and 4 integer types. ^: diff. >>: shift with sign bit as compensate. >>>: shift 32 or 64 bit integer with 0 as compensate.

some assignment statements: <<=, %=, &=

instanceof:

reference instanceof classOrInterface, returns Boolean. depends upon instance not reference, subclass instance is a superclass instance.

the initialization block of for can announce only one type but multiple variables.

continue in while or do while will jump to bool judgement.

Number extended by Byte, Double, Float, Long, Integer, Short. They are classes, can be used like other classes: Double x = 1.5; x.toString(); There are also static methods.

Likewise, there is a Character class.

when needed, automatic convert between object and primitives will apply.

\0 doesn’t mean end of string, it will be counted in length, but can not be printed.

create a formatted string: string s = string.format(“%d end”, 1);

arrays always have a field named length, so simply pass a int[] to a function is enough for it to iterate on this array.

int[] array = new int[] {1,2,3}; //work

for objects, array declaration will only produce null references, no constructor will be called.

date and time: when needed, go to <http://www.runoob.com/java/java-date-time.html>

Regular Expressions: <http://www.runoob.com/java/java-regular-expressions.html>

System.out.println(): System is class, out is a standard object in the class

since there is a difference between objects and primitives, changing an argument will not affect a primitive but will affect an object (except built in classes like Double and Short).

overloading only occurs between methods in the same class.

have access to a method means you can access this method by .method();, not that you can call this method directly like method();

inner blocks can’t have variables of same name like variable hiding in c++.

mutable arguments: at the end of an argument list, use argType… argName to indicate a mutable argument. if additional arguments are presented, they will be collected to an array named argName. Only one mutable argument of one type can be created.

if additional operations need to be taken when destroying an object, write protected void finalize() to do this.

command line io, file io, traverse a directory: <http://www.runoob.com/java/java-files-io.html>

scanner for input: <http://www.runoob.com/java/java-scanner-class..html>

by default, java.lang packet is imported to every java program.

checked = non-runtime and vise-versa

checked exceptions must be handled: if a method will not catch a checked exception, then the method must use throws to declare it. the one who calls this method must do the same: if you call a method which will throw Exception, you must catch it or declare it. you can also use throw to throw an exception.

customized exceptions: checked should inherit exception, runtime should inherit runtimeException.

throw new xxxException(); //throw an exception

even return can’t stop a finally block, only system.exit() can do that

during compiling, only reference type will be checked, superclass a = subclass(); a.supandsubmethod(); can be compiled because method exists in a’s class, namely superclass. a.submethod(); can not be compiled because method doesn’t exist in superclass. during running, jvm assign the type of instance and run its methods.

当使用多态方式调用方法时，首先检查父类(reference type)中是否有该方法，如果没有，则编译错误；如果有，再去调用子类(instance type)的同名方法。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

封装：用setter getter访问。最主要的功能在于我们能修改自己的实现代码，而不用修改那些调用我们代码的程序片段。

any non-abstract class who implement an interface must implement all required methods.

接口文件保存在.java结尾的文件中，文件名使用接口名。

标记接口：最常用，没有任何方法和属性的接口。主要用于以下两种目的：1建立一个公共的父接口；2向一个类添加数据类型：该类通过多态性变成一个接口类型。

包：可以组织为树(不过不知道eclipse中怎么办的)。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。同时也限定了访问权限。如果一个源文件中没有使用包声明，那么其中的类，函数，枚举，注释等将被放在一个无名的包（unnamed package）中。通常使用小写的字母来命名避免与类、接口名字的冲突。同一个包中不用互相import。编译之后的.class文件应该和.java源文件一样，它们放置的目录结构应该跟包的名字对应起来。但是，并不要求.class文件的路径跟相应的.java的路径一样。你可以分开来安排源码和类的目录。

数据结构：看<http://www.runoob.com/java/java-data-structures.html>及后面一页

序列化：该类的所有属性必须是可序列化的。如果有一个属性不是可序列化的，则该属性必须注明是短暂的。

泛型：

所有泛型方法声明都有一个类型参数声明部分（由尖括号分隔），该类型参数声明部分在方法返回类型之前。

每一个类型参数声明部分包含一个或多个类型参数，参数间用逗号隔开。一个类型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。

类型参数能被用来声明返回值类型，并且能作为泛型方法得到的实际参数类型的占位符。（可以用在这个函数的任何地方）

注意类型参数只能代表引用型类型，不能是原始类型（像int,double,char的等）。

泛型类的声明和非泛型类的声明类似，除了在类名后面添加了类型参数声明部分。泛型接口也一样。

泛型类逻辑上是不同类，实际上是一个类。参数类的继承关系不带到泛型类。

当要用泛型类但不知道用什么类泛型时可用通配符：Box<?>表示由？产生的Box泛型。

每个类型参数都能进行修饰和限制，如<T implements List<T>>表示实现了List<T>泛型接口的T类。

java8新特性看<http://www.runoob.com/java/java8-new-features.html>