**Core Java Class Cheatsheet**

* Class can have only *public and default access*.
* Public class needs to be in *same name* java file.
* Single java file can contain more then *one non public class* but can have *only one public class*.
* A public class can be seen by all *classes from all package*
* A class with default access can be seen only by classes within the same package.
* Java file with no public class have no naming restriction.
* *Class can also have final* and *abstract* & *strictfp* non access modifiers.
* An abstract class can not be inistiated.
* A final class *can not be subclassed*.
* A class can not be both *final and abstract*.
* Class visibility can be seen in 3 parameter
  1. If a class can extend another class?
  2. If a class can create instance of another class?
  3. If a class can access methods and variable of another class?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Constructor Cheatsheet**

* Constructor is invoked when a new **object** is ***created.***
* Constructor’s **can** also be *overloaded* but it **can not** be *overridden*.
* Every class has at least one constructor. If user doesn’t provide any JVM will *provide* a default *no arg constructor*.
* *Abstract class* also **has** constructor.
* Constructor *must* have the *same name* as class.
* Constructor can’t have a return type.
* If a method with the same name as class has return type will be treated as normal member method and not constructor.
* Constructor can have ***any access modifier(All)***.
* Default constructor is a no arg constructor which calls the no arg constructor of super class. In case super class doesn’t have any no arg constructor then it will throw a run time exception.
* In case where a class has default constructor, its super class needs to have a no arg constructor.
* *First statement* of a constructor can be either ***this*** or ***super***but can not be both at the same time.
* If coder doesn’t write any *this*  or super call then compiler will add a call to *super.*
* *Instance member*can be accessible only **after** the *super constructor* runs.
* Interfaces do not have constructors.
* *Constructor* are ***not inherited***. Hence can not be *overridden*.
* Constructor can not be directly invoked. It will be invoked(***Implicitly***) when a new object is created or a call by *other constructor*.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Inner Class Cheat sheet for Quick Revision

* ***Inner class***is member of enclosing *class*.
* ***Outer class*** reference is required to initiate inner class.
* Inner class are of 4 type.
* Inner classes defined within method are method local inner class.
* Method local inner class can not access method local variable.
* ***Final*** & ***Abstract*** are the only modifiers available to method local inner class.
* Ananymous inner class don’t have any name.
* Inner classes having ***Static modifier*** are known as *Static inner class*.
* Static nested class can not access non static member of outer class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Access Modifier Cheat sheet for Quick Revision

* ***Public, Private, Protected*** are 3 access modifier
* There are 4 access levels in Java. ***Public, Private, Protected & Default.***
* A class can have only public and default access level.
* Methods and Instance variable (***non local***) can use all 4 access levels.
* If a class is not visible to other class there is no question of accessing member of that class, even when access level of that member is public(***Important***).
* Class visibility should be checked before member visibility.
* If a super class has public member then it will be inherited by subclasses even if it is in other package.
* ***this***always refers to the current executing object.
* A public member can be accessed by all other class even from other package.
* Private members can be accessed only by the code in the same class.
* Default members are not visible to subclasses outside package while protected members are visible to subclassed even when they are in different package.
* Different between *protected*and*default*comes into picture only in the case of subclass outside package.
* Local variables *can not* have *access modifiers.*
* Local variables can have only final non access modifiers that can be applied.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Non Access Modifier Cheat sheet for Quick Revision

* ***Non Access Modifiers***available in Java are *Static, final, abstract, synchronized* & *Volatile*
* Static keyword can be applied to ***Variables****&****Methods.***
* ***Static Variable***  are those variables which are not associated to any instance but it is associated to class means all instances will access the same single copy of variable.
* Local variables *can not*be declared as*Static.*
* *Static keyword can*also be applied to*Methods.*They will work for all the instances and they will not be dependent on instances created.
* Final modifier can be applied to *method* and *variables*.
* Final is the only modifier which will be available to local variable.
* Once declared as final value of the variable can not be changed.
* Final variable don’t get default value opposed to instance variable coz value can’t be changed once assigned.
* Final method can not be overriden.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### String Cheat sheet for Quick Revision

* ***String Object***is used for holding String of text.
* Part of *java.util package.*
* *Strings*can be added by using concatenation Operator(**+**).
* Characters use single quotes(**‘ ‘**) as delimiter while Strings use double quotes(**” “**) as delimiters.
* ***String***objects are immutable. Value of the string object can not be changed once assigned.
* String class is final means method of the String class can not be overriden.
* All ***String literals*** are added to String pool by JVM.
* String have *method* **length()**while Array have a *property* **length** to check the length.
* To overcome the immutable property of String StringBuffer and StringBuilder comes into picture.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Thread Cheatsheet**

* Thread can be created by *extending Thread Class* and overriding the run() method.
* Thread can also be created by *Implementing Runnable interface* and then calling Thread constructor that takes Runnnable argument.
* start() method can be called on Thread Object *only once*.
* Once start() method is called on Thread Object it becomes *thread of execution.*Before start() method called it is said to be in new state and is not considered alive.
* There is no guarantee of order in which Thread will get executed after getting started.
* We can influence the order in which Thread get executed by setting its(Thread) priority(1-10).
* A running thread may enter blocked/waiting state by wait(), sleep or join() call.
* A runnning thread may enter blocked/waiting state because it can’t acquire the lock of a synchronized block of code.
* A dead thread can not be started again.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Static Cheat sheet for Quick Revision

* ***Static***is a Non Access Modifier.
* **Static** modifier can be applied to Variable / Method / Block / Inner Class.
* ***Static members*** belong to class only not any instance.
* Static method **can not** access *instance variable*.
* Static methods **can not** be *overriden*. As they are class specific and doesn’t belong to any Instance.
* Static methods can be ***redefined***.
* Static variable can not be directly accessed by Non Static Methods.
* If a class contain any static block then that block will get executed only when class get loaded in JVM. Creating multiple instances doesn’t mean that Static block will get executed multiple time. Only Constructor will get executed multiple time.
* If Class.forName(“*class\_name*“) is called then static block of the class will get executed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Interface Cheatsheet

* Interface are 100% abstract class(***Implicitly***).
* Interfaces can be implemented by any class from any inheritance tree.
* All methods in Interfaces are abstract.
* Interface can have constants, these constants are public, static and final(***Implicitly***).
* Interface methods are implicitly ***public & abstract.***
* 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 class can extend *more then one* Interface.
* Interfaces can not extend a class or implement an Interface.
* Interface can extend another Interface.
* 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.
  3. It must maintain the exact signature of method.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**String Builder Cheatsheet**

* Difference between***StringBuilder*** and ***String Buffer*** is that StringBuilders method are not synchronized.
* String Builder is ***mutable*** which is opposite to String which is ***Immutable***.
* StringBuilder’s equals method is not overriden, Hence it doesn’t compare the value of StringBuilder. As it does in case of String.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Serialization Cheatsheet**

* ***Serialization*** interface needs to be implemented in order to make object *serialized.*
* Transient instance variable doesn’t serialized with Object state.
* 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 constructor will be invoked.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Overloading Cheatsheet**

* Using the same method name but with *different argument* is called overloading.
* ***Constructors***can also be overloaded
* Overloaded methods must have *different* argument set.
* Overloaded methods ***may*** have ***different return type.***
* Overloaded methods ***may*** have *different access modifier.*
* Overloaded methods ***may*** throw different exception ***broader or narrow*** no restriction
* Methods from **super class**can also be overloaded in subclass.
* Polymorphism applies to overriding not Overloading
* Which overloaded method will be invoked is decided on***compile time*** on the basis of**reference type**.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Overriding Cheatsheet

|  |
| --- |
| * *Constructor*can not be overriden. * Overriding methods must have the same argument set. * Overriden methods must have same return type. These return type can also be the subclass(***covariant return***). * Overriden method **can not** have *more restrictive* access modifier. * Overriden method **can not** throw *new or broader* exception(***Checked***). * Overriden method **can throw** any *unchecked exception.* * **Final** methods ***can not*** be overriden. * ***Private methods*** are *not inherited* to subclass hence it *can not* be *overriden* in subclass. * Polymorphism applies to overriding. * ***Object type*** determines which overridden method will be invoked and that will be decided at the *runtime*.   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

# Exception Cheatsheet

* ***Checked***&***unchecked exception***are two type of Exception***.***
* Checked exception are those exception which are subtype of Exception class but excluding classes which extends ***Runtime exception***.
* Subtype of ***Error*** and ***Runtime Exceptions*** comes under unchecked Exception.
* *Finally* block will **always** be invoked in all condition.
* System.exit() is the only way when finally block will not get executed coz in that case JVM will shut down.
* Custom Exception can also be created by extending Exception class.
* Catch block should be order in the form of most specific to most general. Otherwise compiler will complain for unreachable code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Collection

1. ***Collection***is a data structure in which Objects are stored.
2. Objects can be Added, Deleted and can traversed in Collection.
3. There are 4 type of basic Collection
4. **List :** **Ordered**, **Duplicates** are **allowed**, Indexed
5. **Sets :**May or may not Ordered. **Duplicates** are **not** allowed.
6. **Maps : Duplicate** keys are **not** allowed.
7. **Queue : Ordered**by**FIFO**or**priority.**
8. **ArrayList :**Fast Iteration & Fast Random Access.
9. **Vector:**Synchronized Method.
10. **LinkedList :**Good for implementing Stack and Queue.
11. **HashSet :**Fast Access, No Duplicates, No Ordering.
12. **LinkedHashSet :**No Duplicates, Iterates by insertion order.
13. **TreeSet :**No Duplicates, Iterates in sorted order.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Local Variable Cheatsheet:: Rule for Local Variable

* Can not use any of the access level as it has the life inside method only.
* Only non access modifier that can be applied to Local Variable is Final.
* Local Variables don’t get default value, hence local variable need to be initiaited before it can be used.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Instance Variable Cheatsheet:: Instance Variable

* ***Public, Private, Protected*** all 3 access modifiers can be applied to Instance Variable(**Default** also).
* Instance variable will get default value means instance variable can be used without initializing it. Same is not true for *Local Variable.*
* *Instance Variable*can be marked **final**.
* *Instance Variable* can be marked **transient**.
* *Instance Variable* can not be **abstract**.
* *Instance Variable* can not have **synchroonized** modifier.
* *Instance Variable* can not have **strictfp** modifier.
* *Instance Variable* can not have **native** modifier.
* *Instance Variable* can not have **Static** modifier as it will becomes Class level variable.

|  |  |
| --- | --- |
| **Instance Variable Type** | **Default Value** |
| boolean | false |
| byte | (byte)0 |
| short | (short) 0 |
| int | 0 |
| long | 0L |
| char | u0000 |
| float | 0.0f |
| double | 0.0d |
| Object | null |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Java Inheritance Cheatsheet:: Java Inheritance Rule

* All **public variables** of Super class will be *inherited* by subclass.
* All **default variables** will be inherited by all subclass in the *same package only. Subclass outside the package*will not inherit any default memeber.
* **Private member** *can not* be inherited by subclass because it will not be visible to subclass and hence subclass can create the method or property with the same name without any problem.
* **Protected variable** will be inherited by all subclass in the *same package or outside package***(Different from default)**.
* Methods which are not inherited **can not be overriden**. And hence overriden rules can not be applied to those Methods. But Methods can still be defined in subclass but those methods will not be overriden method but a new method.
* Static methods /variables do not take part in inheritance.
* Even static methods / variables do not take part in inheritance hence they can not be overriden but they can be redefined in subclass. Redefinition is not called overriden but hidden.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*