# Notes:

* String (object):String Pool ,immutable
* StringBuffer (object): mutable

access modifiers:

* public: anywhere in project
* private: over class
* protected: over the package & (subclasses -> between packeges using inheritance)
* default : in the package

* final: keyword -> using with classes & methods
* finally: block -> try... catch , always executed evenif handle Exception
* finalize: method -> garbage collection ( cleanup activity is implemented)

static:

-> method : can't be overridden

-> variable : can't be reintialized

-> class : inner class ->StaticNestedClass

constructor :

* no return datatype , initialize the object state
* constructor name== class name
* can't be marked as final
* when can be private?singleton(one static instance can created form class),factory static method , unitity class-> only contain static methods

super , this ?

* + this:Keyword -> refer to the current instance of class
  + super:keyword ->
    - refer to superclass of current instance
    - call superclass constructor (Default) into subclass constructor (METHOD)

Stack , Heap ?

* + Stack ( LIFO ): local & temporary variables and function call , fast
    - limited size,Managed automatically by the system
  + Heap: dynamic memory allocation using objects ,DS like Arrays&linkedList
    - manage by garbage collector -> slow,Large size

shallow copy , deep copy ?

* + - * Shallow copy : copy refrences to nested objects
        + changchanges in nested objects are reflected
        + in both the original and copied objects.
      * Deep copy :changes in nested objects do not affect each other
        + between the original and copied objects. -> create new object reference

throw , throws?

* + throw : throw Exception if something wrong has occurred
  + throws: the type of Exception like "FileNotFoundException"

composition, aggregation ?

* + composition: relation between objects -> one contain(Own) anthor (object part of anthor )
    - * containing object cannot exist without objects it contains
      * containing object is destroyed ->contained objects are also destroyed.
      * like (car & engine)
  + aggregation: relation between objects -> one contain(Own) anthor
    - * the contained object can exist independently of the containing object
      * like(University & department)

System.out.println()?

* print in the Console ->
* Java.lang package -> System->PrintStream -> static variables like(out),methodslike (println())

\* java 8 ->stream ,lambda ,static&default methods in interface

concepts of oop?

Inhertance:-

* Allows subclasses to inherit behaviors and properties from another class (superclass).
* Enables code reuse and extension without rewriting existing code.

Abstraction:-

* abstract class : at least one abstract method
* -interface: all methods are abstract by default but befor Java-8 default & static methods with implementation
* Hides complex implementation details and shows only essential features of an object
* Methods without implementation

Polymorphism:-

* override ->methods in subclass with a different implementation
* Overloading ->methods with the same name but differnt parameters type or number
* Enables methods to be called on objects of different classes executing different behaviors based on the object type

Encapsulation:-

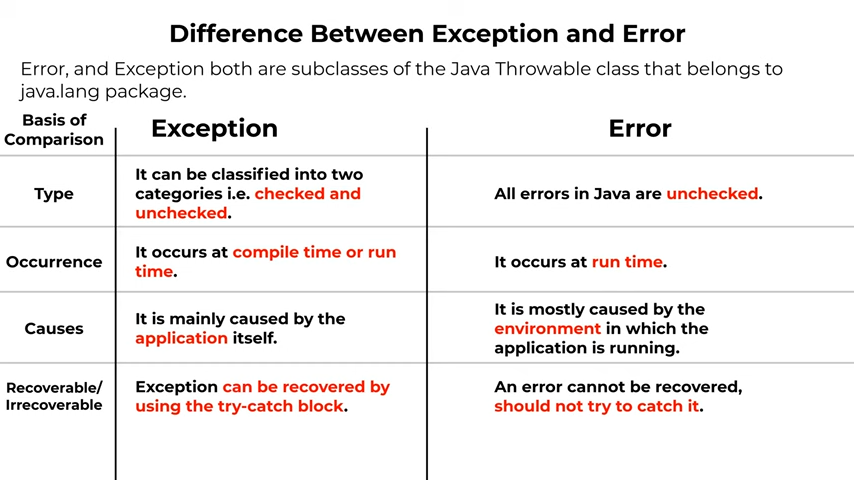
* hide data
* direct access to some of an object's components and protects its internal state.
* private to variables , public to classes (setters & getters)

Exception Handling?...

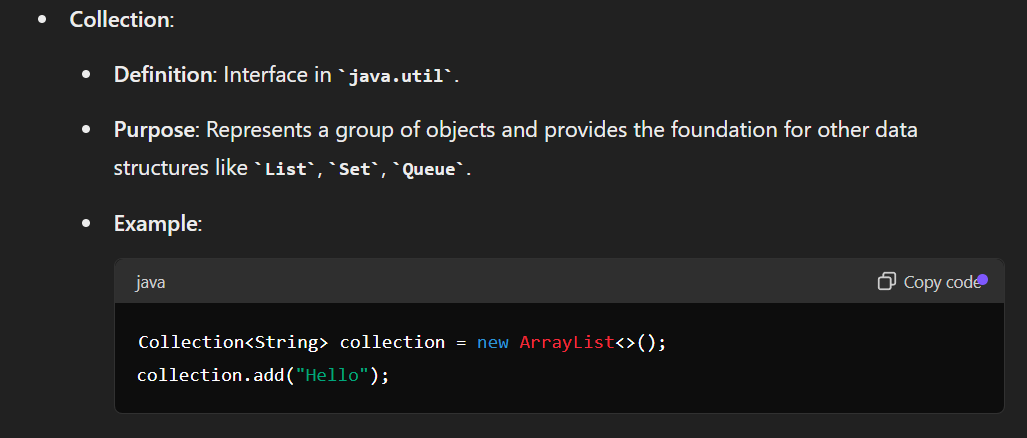
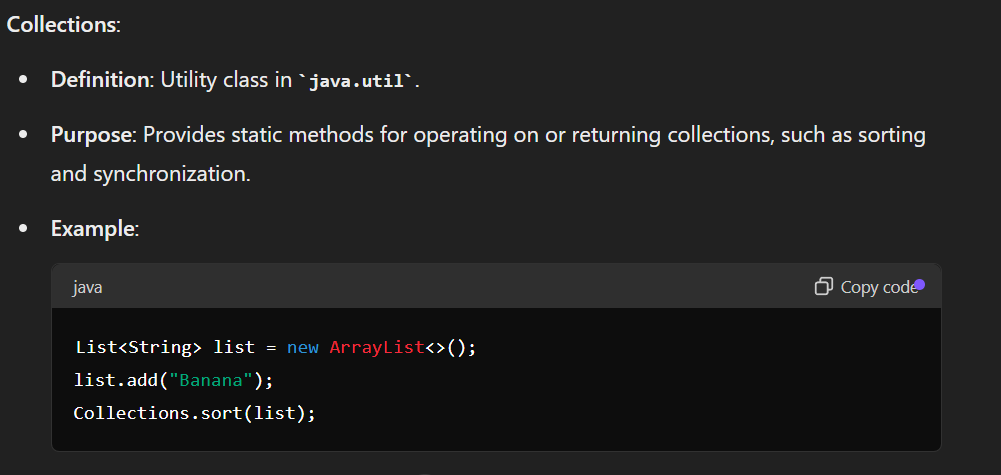
Checked exceptions (Compile-time exceptions)

* + is checked (notified) by the compiler at compilation-time,also called as compile time exceptions.
  + cannot simply be ignored, the programmer should take care of (handle) these exceptions.

Exception & error?



**collection/s** :

* Collection: 
* Collections: 

- array , arraylist , Linkedlist :

* Array:
* ArrayList
* LinkedList

List ,Map, Set

| **Feature** | **List** | **Set** | **Map** |
| --- | --- | --- | --- |
| **Interface** | java.util.List | java.util.Set | java.util.Map |
| **Allows Duplicates** | Yes | No | Keys: No, Values: Yes |
|  |  |  |  |
| **Maintains Order** | Yes | No (unless LinkedHashSet) | Yes (unless HashMap) |
|  |  |  |  |
| **Random Access** | Yes | No | Yes (for keys) |
|  |  |  |  |
| **Access Time** | O(1) for ArrayList, O(n) for LinkedList | O(1) for HashSet, O(log n) for TreeSet | O(1) for HashMap,  O(log n) for TreeMap |
|  |  |  |  |
| **Null Handling** | Allows multiple null elements | Allows a single null element (for HashSet and LinkedHashSet) | Keys: Allows one null key  (for HashMap),  Values: Multiple null values |
|  |  |  |  |
| **Typical Use Cases** | Ordered lists,  frequent read access by index | Unique elements,  fast lookup and deletion | Key-value pairs,  fast lookup by key |
| **Memory Usage** | Depends on implementation, generally lower than Set | Higher memory usage due to storage of elements without duplicates | Higher memory usage due to  storage of key-value pairs |
|  |  |  |  |
| **Performance** | Fast random access, slower insertions/deletions for ArrayList; consistent time for LinkedList | Fast access, insertion, and deletion for HashSet; sorted access for TreeSet | Fast access, insertion,  and deletion for HashMap;  sorted access for TreeMap |

Design pattern:

-creational: singleton ,factory method

-structural: adaptor , proxy , facade

-behavioral: observer , strategy ,command