APIs

Summary:

- `Math..exact`
  - for arithmetic overflow
- Iterator
- Scanner
- String
- StringBuilderr
- Set (HashSet)
- List (Arraylist)
- LinkedList
- Stack
- Queue (LinkedList, PriorityQueue)
- PriorityQueue
- Map (HashMap)

Iterator:

- hasnext
- next
- remove

Scanner:

- findinline(Pattern)
- hasNext..int,byte,pattern()
- next...Int

String:

- charAt
- codePointCount(int begin,int end) returns unicode(ASCII) points within range
- valueOf
- toLowercase()

StringBuilder:

- append()..int,char,etc.
- charAt
- insert(int i, char,int, string...)
- toString

Set(Hashset):

- add
- clear
- clone
- contains
- isEmpty
- iterator
- remove
- size

List (ArrayList):

- add(E e),(int ind, E e)
- clear

- get(int x)
- indexOf
- set(int ind, E e)
- remove

LinkedList (LinkedList):

- add(E e),(int i, E e)
- addFirst
- addLast
- get(int i)
- peek
- remove
- set

Stack (Stack):

- empty()
- peek
- pop
- push
- search(E e)

Queue (LinkedList, PriorityQueue):

- Interface with multiple implementations

PriorityQueue (Binary Heap):

- add(E e)

APIs notes

- iterator
- poll()
- peek()

Map (HashMap):

- put
- size
- get
- contains

Comparator:

- `implements Comparator<T>`
- override compare method

Java Nuances

General Tips:

- Getter and setter
- Override and super
- Java outmatically collects garbage
- &&/|| checks left first
- + strings makes a new string every time, if you want to do in a loop use stringbuilder(reduce memory)
- Everything in Java not explicitly set to something, is initialized to a zero value
  - references (anything that holds an object):null
  - int/short/byte:0
  - float/double:0.0
  - booleans: false.
  - array of something, all entries are also zeroed

Useful built in functions:

- Arrays
  - Arrays.binarySearch(arr, target)
    - Negative value shows where it should be
  - Arrays.sort(arr)

Switch Statement:

- All matching cases will be run unless their is a break statement
- Example

  -
    ```java
    switch (month) {
               case 1: monthString = "January";
                       break;
               case 2: monthString = "February";
                       break;
               case 3: monthString = "March";
                       break;
    ```

```
            case 4: monthString = "April";
                    break;
            case 5: monthString = "May";
                    break;
            case 6: monthString = "June";
                    break;
            case 7: monthString = "July";
                    break;
            case 8: monthString = "August";
                    break;
            case 9: monthString = "September";
                    break;
            case 10: monthString = "October";
                    break;
            case 11: monthString = "November";
                    break;
            case 12: monthString = "December";
                    break;
            default: monthString = "Invalid month";
                    break;
        }
```

Breaking out of for loops:

- if you want to skip a particular iteration, use continue

  -
    ```
    for(int i=0 ; i<5 ; i++){

        if (i==2){

          continue;
        }
    }
    ```

- if you want to break out of the immediate loop use break

  -
    ```
    for(int i=0 ; i<5 ; i++){

        if (i==2){

            break;
        }
    }
    ```

- if there are 2 loop, outer and inner.... and you want to break out of both the loop from the inner loop, use break with label

-
```
lab1: for(int j=0 ; j<5 ; j++){
     for(int i=0 ; i<5 ; i++){

        if (i==2){

          break lab1;
        }
     }
}
```

Generics:

- Definition
  - generics are a facility of generic programming
    - a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters
  - ex: compiletime: List¡String¿ runtime:List
- Notes
  - in java, generics are only checked at compile time for type correctness
  - generic type information is then removed via a process called type erasure, to maintain compatibility with old JVM implementations, making it unavailable at runtime
- Sources
  - `https://en.wikipedia.org/wiki/Generics_in_Java`

Type Classifications:

- Concrete Types
  - concrete types describe object implementations, including memory layout and the code executed upon method invocation
  - the exact class of which an object is an instance not the more general set of the class and its subclasses
  - beware of falling into the trap of thinking that all concrete types are single classes!
  - Set of Exact Classes
  - ex: List x has concrete type ArrayList, LinkedList, ...
- Abstract Types
  - Abstract types, on the other hand, describe properties of objects

  - They do not distinguish between different implementations of the same behavior
  - Java provides abstract types in the form of interfaces, which list the fields and operations that implementations must support

Access Modifiers:

- public
  - any class can access
  - accessible by entire application
- private
  - only accessible within the class
- protected
  - allow the class itself to access them
  - classes inside of the same package to access them
  - subclasses of that class to access them
- package protected
  - default
  - the same class and any class in the same package has access
  - protected minus the subclass unless subclass is in package
- Static: Belongs to class not an instance of the class

Things to override in new object (for hashing and equality uses):

- public int hashCode()
- public boolean equals(Object object)

```
ex: Tiger
   @Override
   public boolean equals(Object object) {
      boolean result = false;
      if (object == null || object.getClass() != getClass()) {
         result = false;
      } else {
         Tiger tiger = (Tiger) object;
         if (this.color == tiger.getColor()
               && this.stripePattern == tiger.getStripePattern()) {
```

```
            result = true;
        }
    }
    return result;
}
```

Sources:

- https://www.cs.utexas.edu/~scottm/cs307/codingSamples.htm