General

Xor = ^
Floor division = Math.floorDiv(a,b) or Math.floor(a/b);
|x| = Math.abs(x)
Max number = Math.max(a,b);
Min number = Math.min(a,b);
A^b = Math.pow(a,b);
Round x = Math.round(x);

Converting string to boolean = Boolean.parseBoolean(string s);

Java automatically in and out boxes primitves into wrapper objects.
Random float from with 0.0 to and without 1.0: Math.random();
Random integer between min and max: `Min + (int)(Math.random() * ((Max - Min) + 1))`
Random integer between 0 and n: (new Random()).nextInt(int bound);
Random Boolean: (new Random()).nextBoolean();

Sa

To use super class method we write super.[methodname]()
Subclasses must have super in the constructor that passes values to the constructor of the superclass
Double/long can't be converted to float/int and needs to be casted to float (kanskje feil)

Classes

Java.uti.scanner

New Scanner(X x); X can be File for reading files or System.in for taking output from console
.nextLine(); returns String stripped of \n
.nextInt(); returns next integer
.hasNext(); returns true if next line exists; use for files

File

New File(String pathname);

Array

Initilized with:

```
type[] x = {};
type[] x = new type[size]
```

All arrays have instance variable length which contains integer length of array

.copyOfRange(T[] original, int from, int to); returns array of T objects from index from to index to including the elements at these indexes
.copyOf(T[] original, int newLength); returns a new array with same values as original but a new size, all new values are null, if newLength is smaller then elements at the end are removed
.asList(T[] arr); returns a Collections implementation of the array which allows access to all Collections API.

ArrayList<E>

Intilized with new ArrayList<>(); or it can be cotructed with a container that has the collections api or an int argument for size
Int size(); returns number of elements in the list including nulls
Boolean isEmpty(); returns true if empty
Boolean contains(Object o); returns true if arraylist contains object.
Int indexOf(Object o); returns index of an object or -1 if list doesn't cntain object
Int lastIndexOf(Object); returns index of last occurrence of object
Object[] toArray(); returns array of the object.
E get(int index); returns element at index.
E set(int index, E x); replaces the element at index with x and returns the element
Boolean add(E x); add x to end of arraylist
Void add(int index, E x); adds x at index and shftes everything including the old element at index to the right.
E remove(int ind); removes element at index ind and returns it
Boolean remove(Object o); removes first ovurance of o and returns false if not found
List<E> sublist(int from,int to); returns a smaller version of the same array from and with index to and without index to, changes in the sublist are changes to the arraylsit and viceversa

HashMap<K,V>

Constructed with new HashMap<K,V>();
Boolean containsKey(Object key); returns true if hashmap contains the key
Boolean containsValue(Object key); returns true if hashmap contains the Value
V get(Object key); returns the value of the key
V put(K key, V value); creates a new entry, returns the old value if key already exists or null
Set<K> keySet(); returns set of keys
V remove(Object key); removes the key and its value and returns the value
Boolean remove(Object key, Object value); removes the key and value only if the key is mapped to the value
Replace(K k, V old, V ny); replaces the old value of k with the new value
Int size(); returns number of keys
Collections<V> values(); retuns a collection of all values

Integer (wrapper)

New Integer(int x);
Static parseInt(String s); returns int of string

Boolean (wrapper)

New Boolean(boolean x);
Static parseBoolean(String s); returns boolean of string ignoring case

String

.compareTo(String x); compares string lexographically (b>a)
.replaceAll(String s, String x); returns String with all occurrences of s replaced with x
.replaceFirst(String s,String x); same as the above but only for the first occurrence of s
.split(String x); returns String array that results from splitting the string around x removing x from the resulting strings;
.contains(String
.trim(); returns string with all leading and trailing space and \n removed
.indexOf(String s); returns the index of the first character of the first occurrence of the string
.toUpperCase(); returns string converted to upper case
.toLowerCase(); returns string converted to lower case
.substring(int start, int end);returns string from index start to index end of string
.endsWith(String suffix); returns true if string ends with suffix
.length(); returns length of string as int

Exceptions

Thwoables which terminate a program are divided into two subcleasses, errors and exceptions
Errors like jvm errors:
Exceptions which are further divided into two
1. checked exceeptions like compiletime errors
2. unchecked exceptions like nullpointer exceptions and RT exceptions
3. user defined exceptions

To create custom exceptions we create a new exception class that extends RuntimeException the add a constructor that passes the error message to the super constructor.
To throw that exception we write throw new [CUSTOM EXCEPTION]([ARGS]);

Exception checking is done with throws and try-catch-finally
Adding throws next to the method name makes the jvm ignore the exception thrown by that method and continue running
Try catch runs code in the try section, then runs the catch section if an exception of the specified type occurs then runs the code in the finally section regardless if the exception got resolved

```
try{} catch (Exception e){} finally{}
```

Exception can be any subclass of Exception
Some Exceptions in java:
NullPointerException OutOfBoundsException

Interfaces

Compareble<T>:
@Override
public int compareTo(T o) {return x}
x=0 if o==this, x>0 if this > o, x<0 if this<o

Iterator<T>
boolean hasNext(); returns true if next element exists
T next(); returns the next element in iteration
Allows looping over elements with while(iterator.hasnext()){T x = iterator.next()}.
Is implemented by creating an inner class that implements iterator then making the upper class implement Iterable<T> then adding a method that creates an object of the inner class in the full class

Iterable<T>
Iterator; returns iterator
Allows for loops with for(T x:this)

Threads

Initilization:

1. Make the class implement the Runnable interface which has the run method which contains all code to be run in parallel for the object. Create an object and pass it to the constructor of a thread as a Runnable type. Call the .start() method on the Thread object

2. make the class extend the Thread class and override the void run() method. Call start on the object

Main thread is always initialized automatically and an eventdispatcher thread is initialized if program has gui

Thread termination:

A Thread is automatically terminated when it has no code to be run in the run() method.
.interrupt(); tells the thread to stop as soon as possible

Thread waiting:

.sleep(int ms); tells the thread to wait for int milliseconds
.join(); called from within a thread on other threads to make the thread wait until other threads are done

Calling a method locked by another thread; wait until the other thread unlocks the method or signalled
.CountDownLatch.await(); makes the thread callin it await until the countdown latch has counted to zero or the time given as argument has passed.
All those methods throw InterruptedException and should be surrounded with a tray catch and finally if the thread might have a lock

Thread:

Constructed with new Thread(runnable r) or without a runnable or with a runnable and string.
Void .start(); starts the thread which makes it call its run method.
Static Thread currentThread(); returns current active thread i.e. the thread calling the method
Static int activeCount(); returns number of active threads
Void interrupt(); interrupts the thread.
Void join(). Makes the thread calling it wait until the thread its being called terminates.
Thread1.join() in main makes main wait for thread1.
static void sleep(long millis) throws InterruptedException; makes the thread calling the method wait

Condition:

A condition is an interface of the lock object that allows passive waiting and a separate lock unlock flow of the main lock object
For example we if we have produces and consumer we can add a condtion for consumers where if there isn't enough element we call condition.await(); when the producer adds an element we make it call condition.signal() which awakes the thread that called await on the condition.
When a thread calls await on the condition The lock associated with this Condition is atomically released and the current thread becomes disabled for thread scheduling purposes and lies dormant until one of four things happens: signal() is called, signalAll() is called, thread is interrupted, or superios wake up happens
After the condition signal is called, the thread that was awaiting needs to acquire a lock again. In all cases, before this method can return the current thread must re-acquire the lock associated with this condition. When the thread returns it is guaranteed to hold this lock.

Constructor: a condition is created with Condition c = [LOCK object].newCondition();
Void await() throws InterruptedException; causes the thread to await until the signal method is called or interrupted.
Void signal(); signals to one thread awaiting to start
Void signalAll(); signals to all threads to start

GUI

All relevant GUI components are a subclass of Component.
The main component of a gui is the JFrame window.

This is initialized by creating a Jframe object with JFrame(String title); and calling [.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); .setVisible(true);] on it

The next component of a gui is the JPanel which is a container for components.
It's better to have this before declaring the window object and all gui elements

```
try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
```

Call .pack() on the fram object after all components added or when a new component is added at RT

Classes:

JFrame:

.setLayout(LayoutManager m); sets the layout of the frame;
.add(Component c); adds component to gui

JPanel:

L

Jlabel:

Is either intilized with text or intilized with nothing and text is added later with .setText(String s);

JButton:

Constructed with no arguments or string for text inside button.
.setText(String s); changes string in button
.addActionListener(ActionListener l); adds action listener which calls its method actionPerformed(); when the button is clicked.
Can be declared with anonymous inner type by:

```
.addActionListener(new ActionListener() {
            @Override public void actionPerformed(ActionEvent e){} });
```

Data structures

Hashmap:

A fixed size array of nodes where index of node is determined by key.hashcode()%arr.length

Dynamic Array:

A container of a fixed array which creates new array with same size and elements at indexes when there isn't enough space
Note: java doesn't allow generic arrays (E[]) and creating an array of objects and casting it to (E[]) is right but unsafe

Linked List:

LIFO: Stack;
cases: empty list, removing the only element,

implementation: replace next of last node with new node for addition and replace the next of node before last node with null for removal

FIFO: Queue
Cases: empty list, removing the only element
Implementation: replace first node with next node for removal and add new node to the next of the last node

Priority Queue: sorted list;
Cases: empty list; list has one node, new element bigger than first, new element bigger than last, else(element in middle)
Implementation:

Indexed List:
Cases: empty list, list with one element, invalid index, index at beginning, index at end, index in middle
Implementation:

Adding: if empty, if ind 0, if ind= size, else(middle)
Removing: if empty, if ind=0, if ind=size-1, else(middle)