Assignment 5

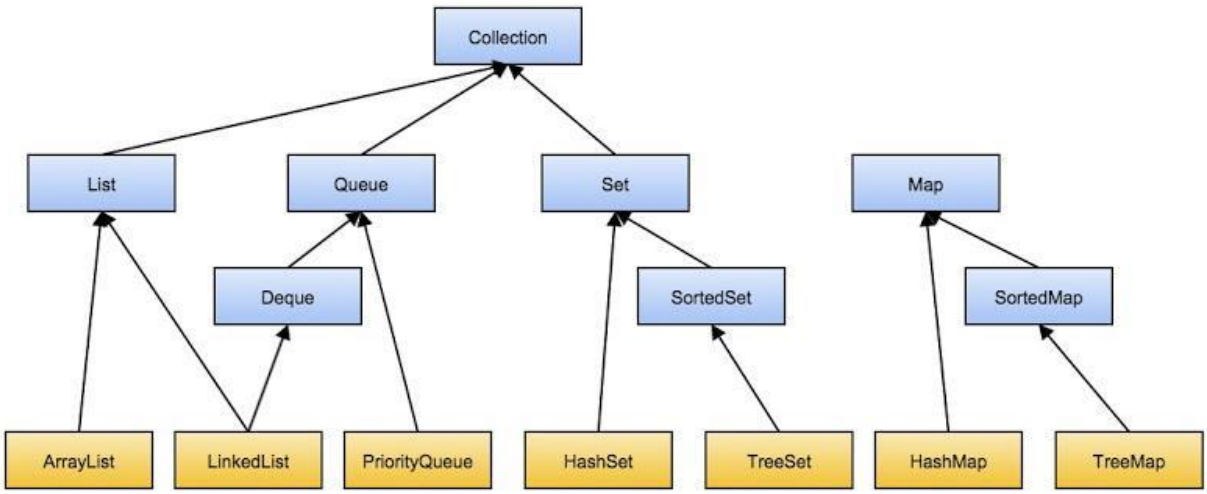
CompletableFuture is used for asynchronous programming in Java. Asynchronous programming is a means of writing non-blocking code by running a task on a separate thread than the main application thread and notifying the main thread about its progress, completion or failure.

A Future is used as a reference to the result of an asynchronous computation. It provides an isDone() method to check whether the computation is done or not, and a get() method to retrieve the result of the computation when it is done.

Basic Data Types
primitive type
    byte, short, int, long, float, double, char, boolean
wrapper class
    Byte, Short, Integer, Long, Float, Double, Character, Boolean

String/StringBuilder/StringBuffer
String
    immutable
    thread safe
StringBuilder
    mutable
    not thread safe
StringBuffer
    mutable
    thread safe

Collection



List

ArrayList

address 0

data0	data1	data2	data4	data5	

O(1)

LinkedList

node1 data next = node2				node3 data next node4
		node2 data next = node3		

O(n)

vector:

thread safe

Stack

thread safe,

FILO

push, pop

Deque: ArrayDeque

first [ ] last

replace Stack: deque.offerFirst(), deque.pollFirst();

Set

HashSet

unique

don't’ keep insertion order

TreeSet

unique
sorted

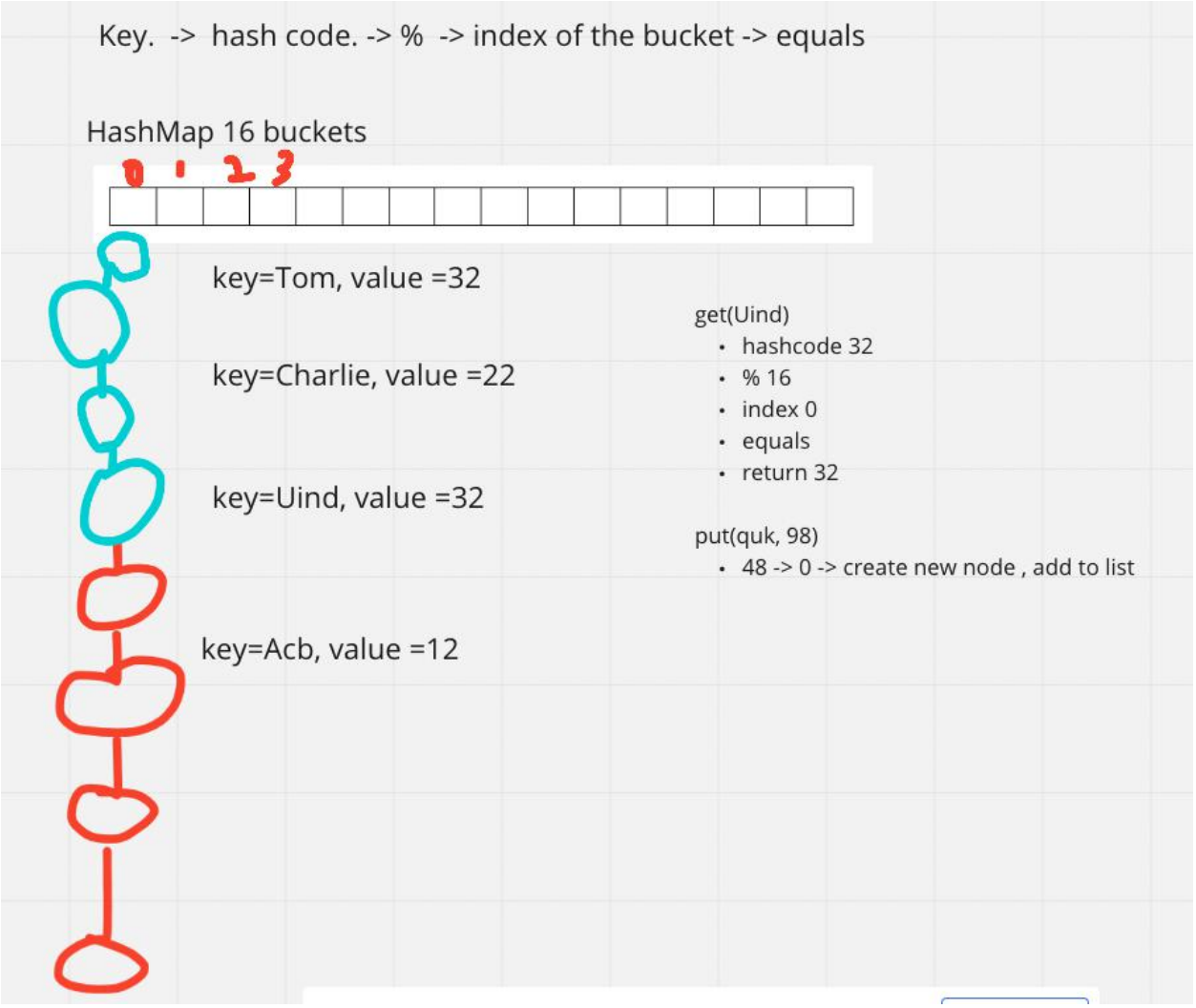
LinkedHashSet

unique
keep insertion order

Map

HashMap
LinkedHashMap
TreeMap
HashTable
ConcurrentHashMap

HashMap



Queue

FIFO

Heap

PriorityQueue

minHeap
maxHeap

array

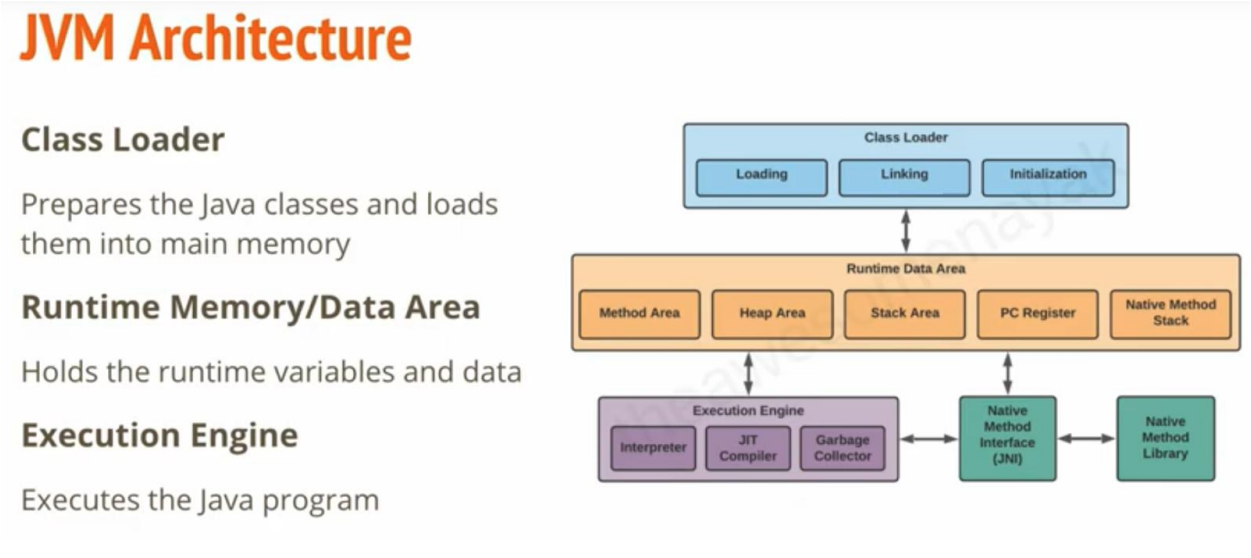
int[] String[] Object[]
int[][], char[][]

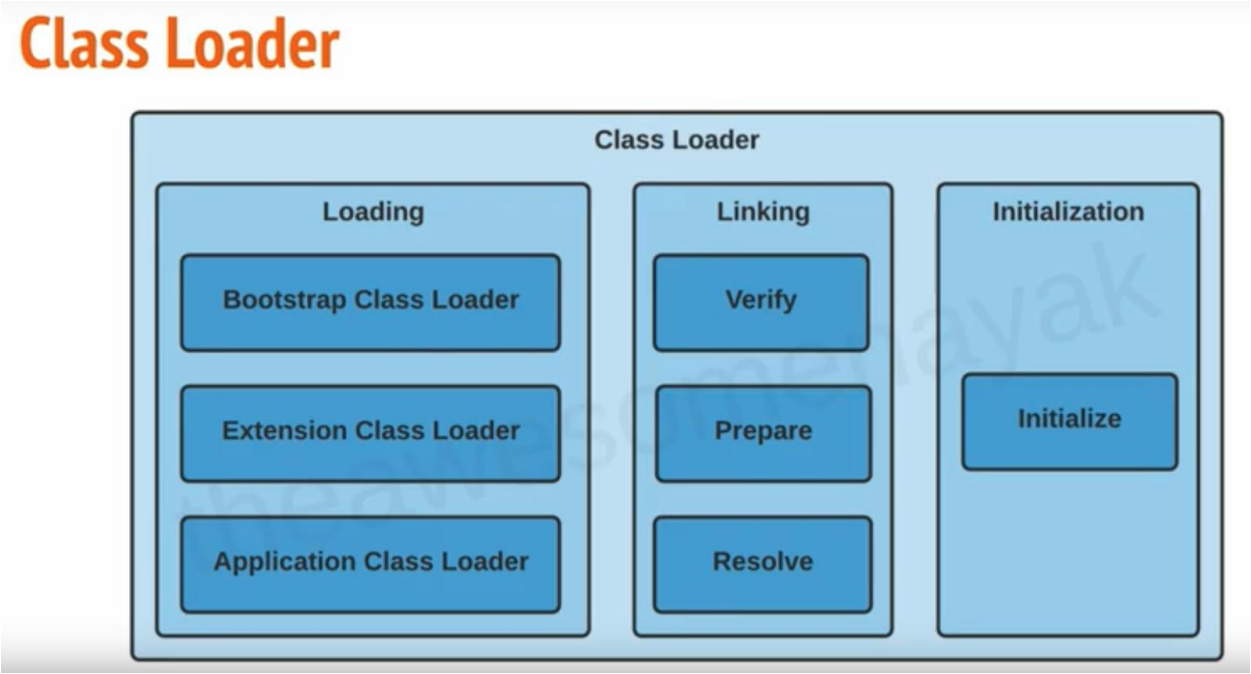
list vs set

HashMap vs HashTable vs ConcurrentHashMap

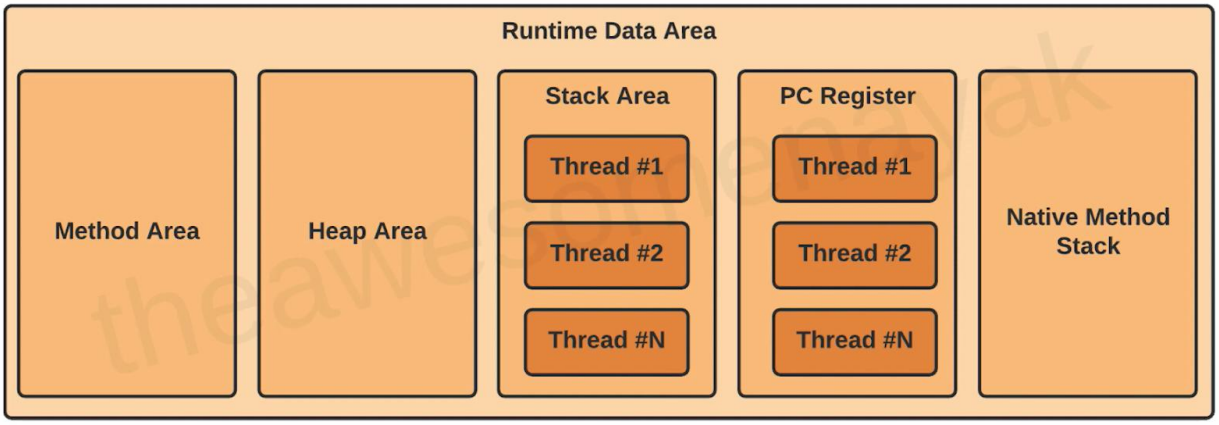
HashSet <- HashMap

JVM

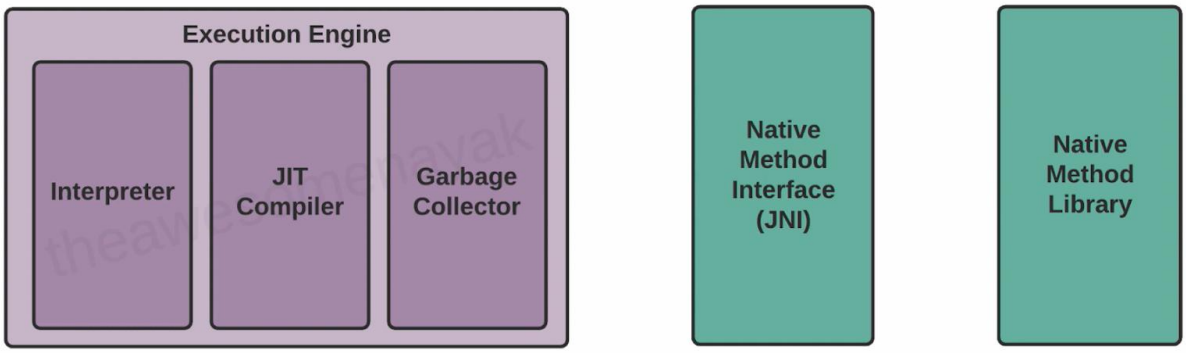




Runtime Data Area



Execution Engine

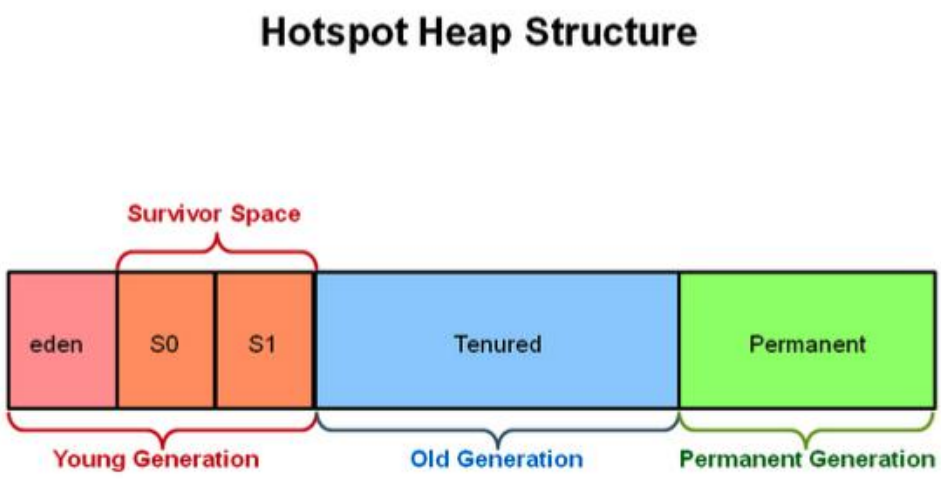


Garbage Collector
- serial GC
- parallel GC
- G1 GC

chunk1 rank2	chunk2 rank1	chunk3 rank3	

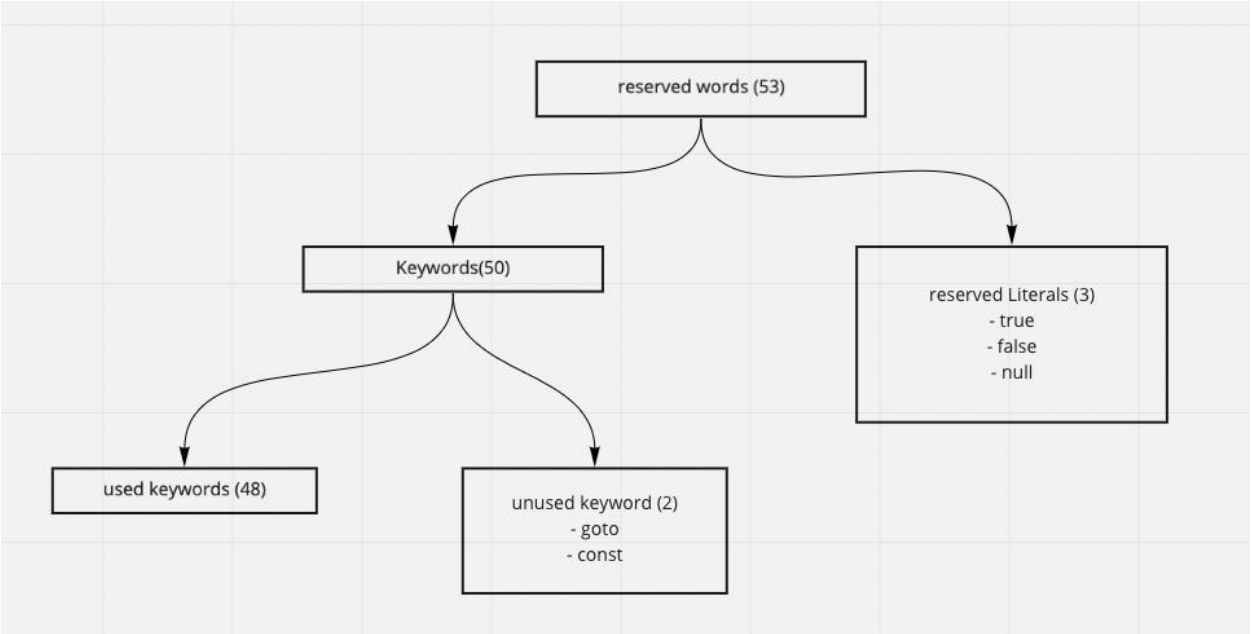
CMS GC(Concurrent Mark Sweep) G1
- deprecated since java 9

completely removed in java 14



minor GC
major GC

Keywords



for data types
byte, short, int, long, float, double, char, boolean
flow control
if, else, switch, case, default, for, do, while, break, continue, return
modifiers
public, private, protected, static, final, abstract, synchronized, native, strictfp,
transient, volatile
exception handling

try, catch, finally, throw, throws, assert

class related

class, package, import, extends, implements, interface

Object related keywords,

new, instanceof, super, this

Final

final variable

create constant variable

must be initialized

```
public class FinalKeyword {
    public static void main(String[] args) {
        final int a = 2;
//        a = 3; // compile error
        final List<Integer> list = new ArrayList<>();
        list.add(1);
//        list = new ArrayList<>(); // compile error


    }
}
```

final method

can’t be overridden

```
class A {
    final void method() {
        System.out.println("134");
    }
}

class B extends A {
//    @Override
//    void method() {
//
//    }
}
```

final class

can’t be extended

```
final class C {}
//class D extends C {}; // compile error
```

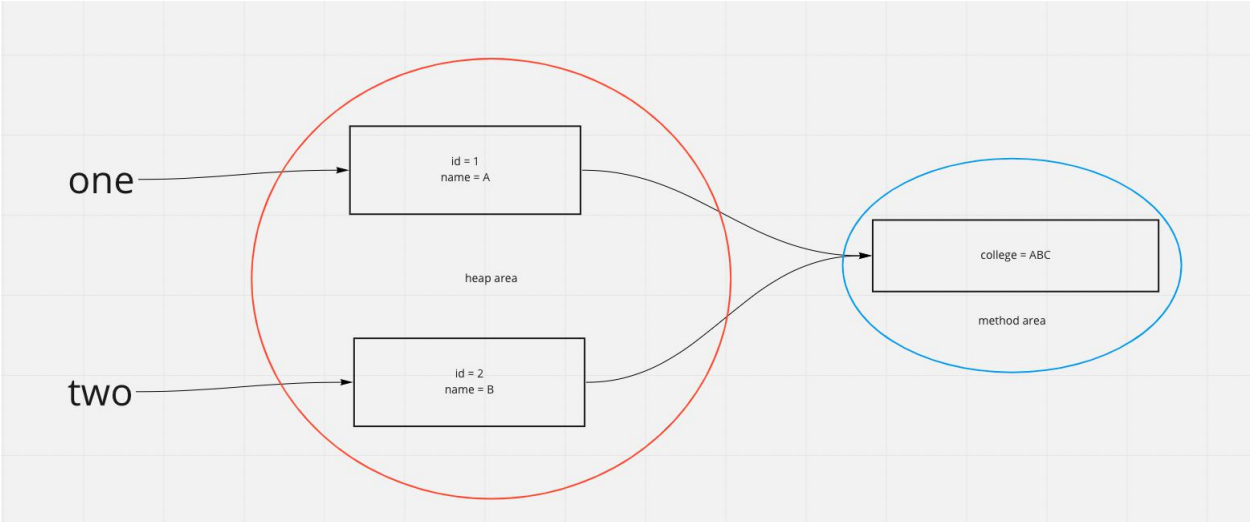
immutable class

final class
private final fields
no setter
return deep copy of the collections for getter

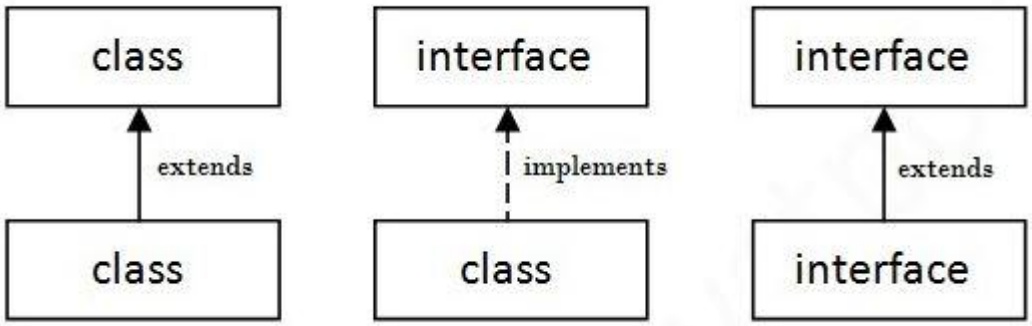
```
final class ImmuClass {
    private final int id;
    private final String name;
    private final Map<Integer, Integer> map;

    ImmuClass(int id, String name) {
        this.id = id;
        this.name = name;
        map = new HashMap<>();
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public Map<Integer, Integer> getMap() {
        Map<Integer, Integer> newMap = new HashMap<>();
        for (Map.Entry<Integer, Integer> entry: map.entrySet()) {
            newMap.put(entry.getKey(), entry.getValue());
        }
        return newMap;
    }
}
```

static

- block
- variable
- methods
- classes



implements vs extends



OOP
Abstraction
Abstract class
interface

Encapsulation
declare all variables be private
declare setter and getter

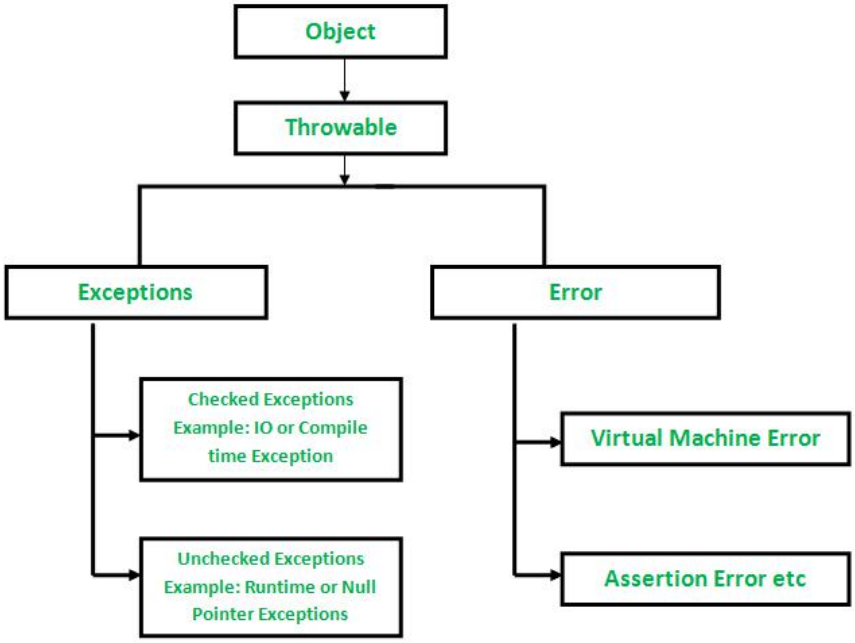
Inheritance
extends
implements

Polymorphism

override
overload

```java
class POL {
    public void method1() {
    }
}

class IKH extends POL {
    @Override
    public void method1() {

    }

    public void method1(int i) {

    }

}
```

Exception



checked exception vs unchecked exception

how to handle the exception

try catch
Throws
how to customize exception

```
class UserNotFoundException extends Exception {
    public UserNotFoundException() {
        super();
    }

    public UserNotFoundException(String msg) {
        System.out.println(msg);
    }
}
```

How to handle multiple exception

```
try {
// business log
} catch (IOException ioe) {

} catch (SQLException sqle) {

} catch () {}
catch () {}
catch () {}
```

—----------

```
try {
} catch (IOException | SQLException | ….    ex) {

}
try {
Connection con = DataDreiver.getConnection();
} catch(IOException ioe) {

} catch (Exception ex) {

} finally {
        if (con != null) con.close();
}
```

Generics

easier and less error prone
enforce type correctness at compile time
without causing any extra overhead to your application

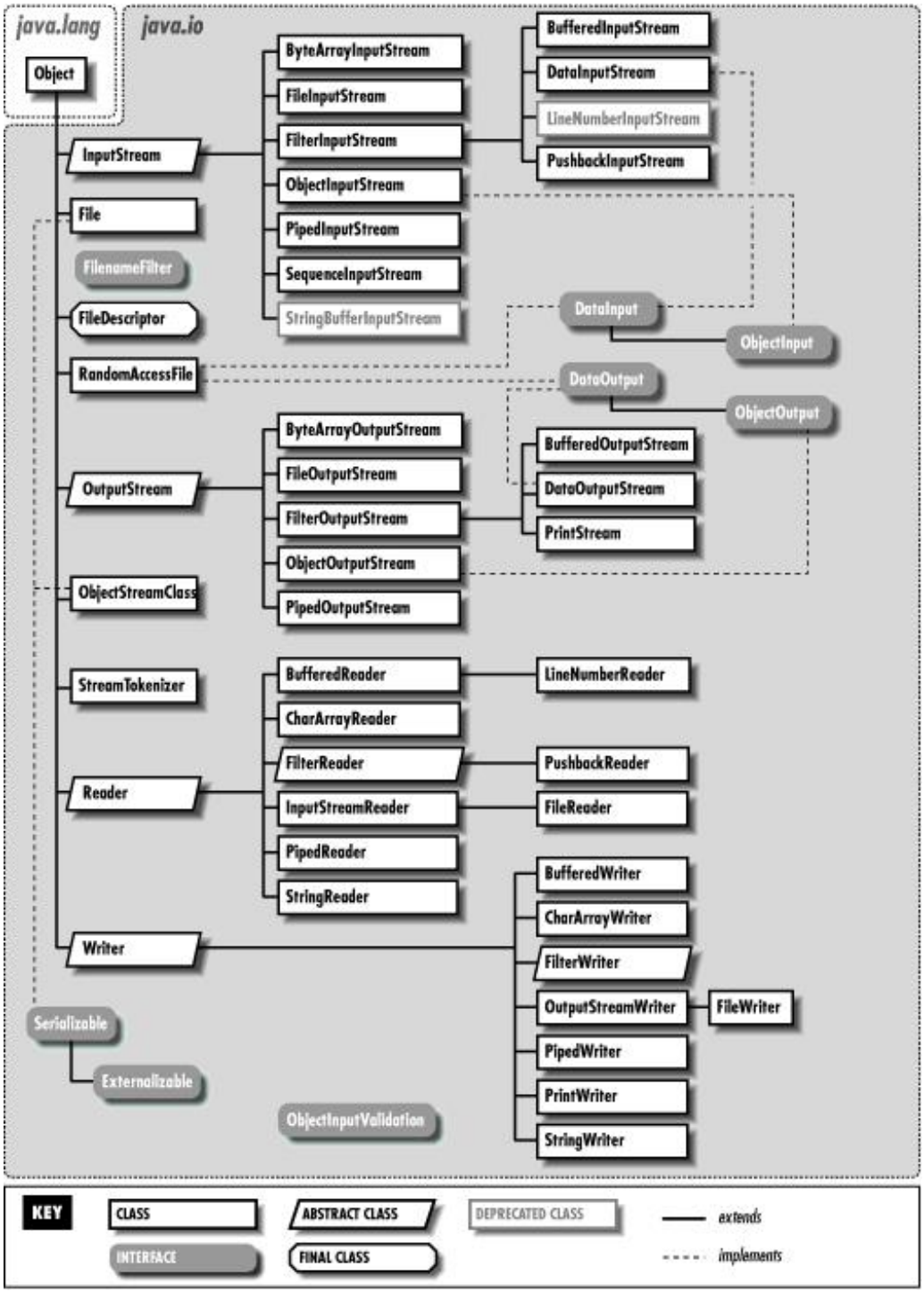
```java
class GenNode<K, V> {
    K key;
    V value;

    public GenNode(K key, V value) {
        this.key = key;
        this.value = value;
    }
}
```

```java
public static <E> E getFirstElements(E[] arr) {
    return arr[0];
}

public static <E, U> E getFirstElements(E[] arr1, U[] arr2) {
    return arr1[0];
}
```

<? extends E>
<? super T>
<T extends E>

IO stream

Stream
    a continuous flow of data
Byte Stream
    InputSteam, OutputStream
    1 byte = 8 bits
CharaterStream
    Reader, Writer
    2 byte = 16 bits



File

the File class is part of java.io

give you access to underlying file systems

Serialization and deserialization

```java
import java.io.Serializable;

public class Student implements Serializable {

    long serializedUId = 12345556;

    private String name;
    private int age;
    private transient int ssn;

    public int getSsn() {
        return ssn;
    }

    public void setSsn(int ssn) {
        this.ssn = ssn;
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

```java
public class SerDemo {

    public static void main(String[] args) {
        Student student = new Student( name: "Charlie",  age: 12);
        student.setSsn(123456);
        try {
            OutputStream fileout = new FileOutputStream( name: "/Users/shaohua/Desktop/JavaMaterial/student.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileout);
            out.writeObject(student);
            out.close();
            fileout.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class DeserDemo {
    public static void main(String[] args) {
        Student student = null;
        try {
            InputStream fileIn = new FileInputStream( name: "/Users/shaohua/Desktop/JavaMaterial/student.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            student = (Student) in.readObject();
            in.close();
            fileIn.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        System.out.println(student.getAge());
        System.out.println(student.getName());
        System.out.println(student.getSsn());


    }
}
```

Java 8 features

lambda

functional programming

less code

(arguments) -> {body}

```java
public class JavaNewFeatures {
    public static void main(String[] args) {
        Drawable d = () -> {
            System.out.println("hello, drawing a circle");
        };
        d.draw();


        Draw d2 = new Draw();
        d2.draw();


        Queue<Integer> maxHeap = new PriorityQueue<>((e1, e2) -> e2-e1);
    }
}

interface Drawable {
    public void draw();
}

class Draw implements Drawable {
    @Override
    public void draw() {
        System.out.println("hello, drawing a circle");
    }
}
```

Functional Interface

- Predicate
  - public boolean test(T t)
- Function
  - public R apply(T t)
- Consumer
  - public void accept(T t)
- Supplier
  - public R get()

```java
public class JavaNewFeatures {
    public static void main(String[] args) {

        Supplier<Double> generateRandomNumber = () -> Math.random();
        System.out.println(generateRandomNumber.get());

    }
}

@FunctionalInterface
interface SayBey {
    void sayBye();

    default public void sayHello() {
        System.out.println("hello");
    }

    default public void sayGM() {
        System.out.println("good morning");
    }
}
```

Optional

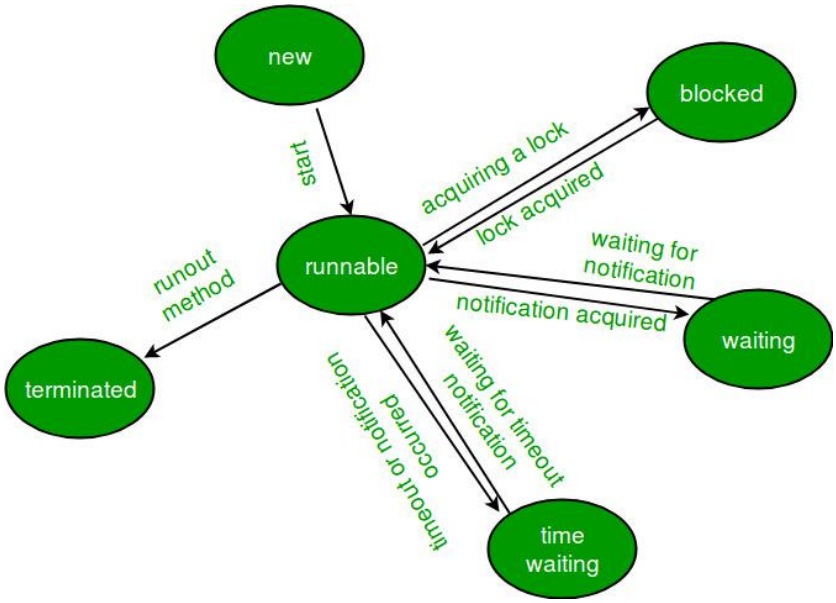
```
if (obj == null) {
…
} else {
…
```

```java
public static void main(String[] args) {

    String str = "abf";
    if (str == null) {
        System.out.println("nothing here");
    } else {
        System.out.println(str);
    }

     Optional<String> opt = Optional.ofNullable(str);
     System.out.println(opt.orElse( other: "Nothing here"));

 }
```
}
Stream API
intermediate operation: return a stream as result
map, flatmap, filter…
terminal operation: return nun-stream
forEach, collect …

```java
public static void main(String[] args) {

  List<Integer> list = new ArrayList<>(Arrays.asList(1,2,3,4,5,6,7,8,9));
  System.out.println(list.stream().filter(e -> e>3).collect(Collectors.toList()));

 }
```

Mutli-threading
thread vs process
Process
independent memory space, heap, OS resources
thread
shared memory space
private stack, program counter, register
Thread states
new
thread create, not yet start
runnable
executing in JVM
blocked
wait for a monitor lock to enter synchronized block or method
waiting
Object.wait with no timeout
Thread.join() with no timeout
park()
timed_waiting

thread sleep
Object.wait() with timeout
thread.join with timeout
park
terminated
thread has completed

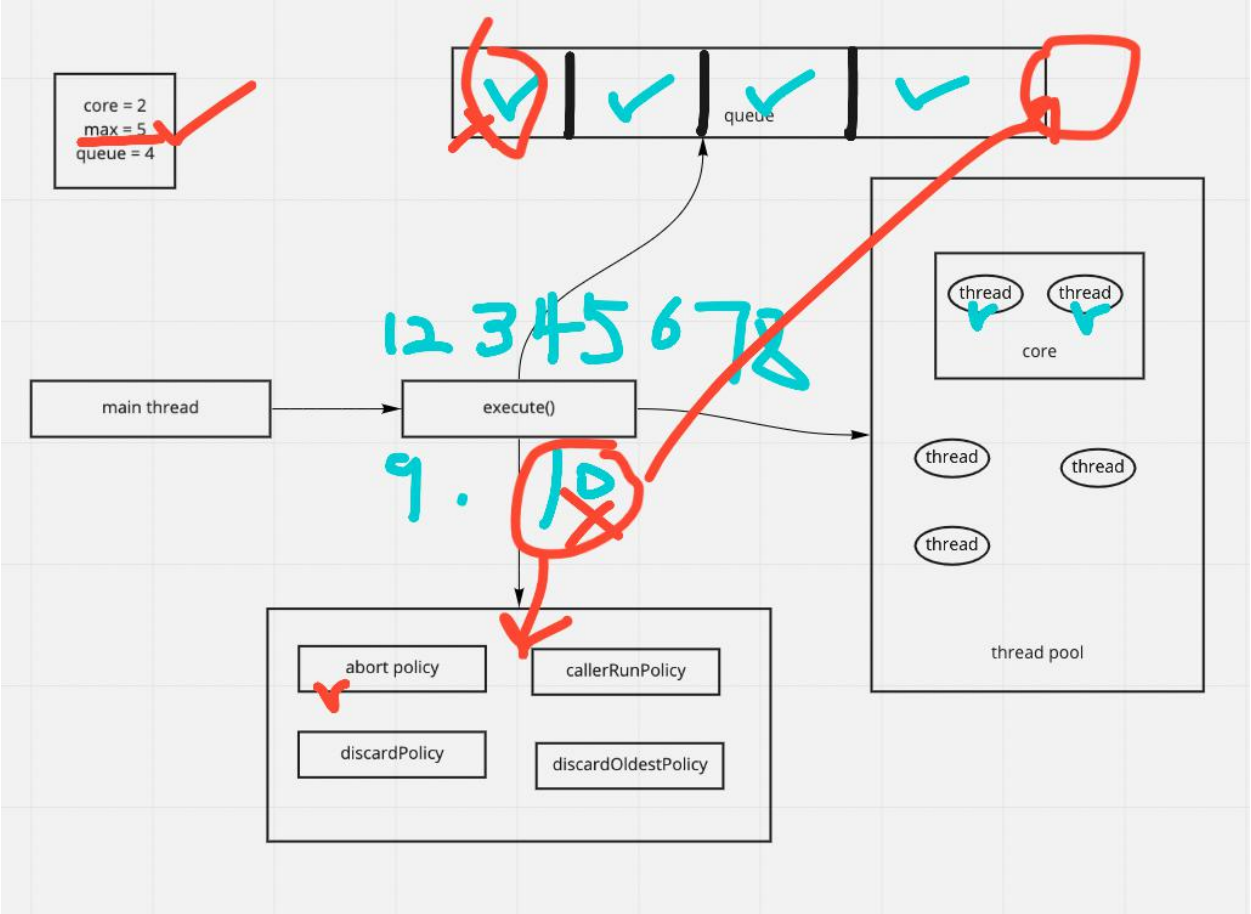


thread creation
extends Thread
implements Runnable
implements Callable
thread pool
runnable vs callable
no return / has
no exception / has
run() / call()

Thread pool
customized thread pool

ThreadPoolExecutor
corePoolSize
maximumPoolSize
KeepAliveTime
Time unit
work queue
thread factory
handler
abortPolicy

callerRunPolicy
discardPolicy
discardOldestPolicy


core = 2
max = 5
queue = 4
queue
thread
thread
core
thread
thread
thread
thread pool
main thread
execute()
abort policy
callerRunPolicy
discardPolicy
discardOldestPolicy

```java
public class ThreadPoolDemo {
    public static void main(String[] args) {
        ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(
                corePoolSize: 2,
                maximumPoolSize: 5,
                keepAliveTime: 2L,
                TimeUnit.SECONDS,
                new ArrayBlockingQueue<>( capacity: 4),
                Executors.defaultThreadFactory(),
                new ThreadPoolExecutor.AbortPolicy()
        );

        for (int i=1; i<=10; i++) {
            threadPoolExecutor.execute(
                    () -> {
                        System.out.println(Thread.currentThread().getName() + " is working");


                    }
            );
        }

        threadPoolExecutor.shutdown();
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-11.0.14.jdk/Contents/Home/bin/java -javaagent:/Appli
Exception in thread "main" java.util.concurrent.RejectedExecutionException Create breakpoint :
    at day5.ThreadPoolDemo.main(ThreadPoolDemo.java:21)
pool-1-thread-2 is working
pool-1-thread-5 is working
pool-1-thread-1 is working
pool-1-thread-1 is working
pool-1-thread-1 is working
pool-1-thread-4 is working
pool-1-thread-3 is working
pool-1-thread-5 is working
pool-1-thread-2 is working
```

in-built thread pool

```java
public static ExecutorService newFixedThreadPool(int nThreads) {
    return new ThreadPoolExecutor(nThreads, nThreads,
                                  keepAliveTime: 0L, TimeUnit.MILLISECONDS,
                                  new LinkedBlockingQueue<Runnable>());
}
```

```java
@NotNull
public static ExecutorService newSingleThreadExecutor() {
    return new FinalizableDelegatedExecutorService
        (new ThreadPoolExecutor( corePoolSize: 1,  maximumPoolSize: 1,
                                 keepAliveTime: 0L, TimeUnit.MILLISECONDS,
                                 new LinkedBlockingQueue<Runnable>()));
}
```

```java
@NotNull
public static ExecutorService newCachedThreadPool() {
    return new ThreadPoolExecutor( corePoolSize: 0, Integer.MAX_VALUE,
                                   keepAliveTime: 60L, TimeUnit.SECONDS,
                                   new SynchronousQueue<Runnable>());
}
```

```java
public ScheduledThreadPoolExecutor( @Range(from = 0, to = java.lang.Integer.MAX_VALUE) int corePoolSize) {
    super(corePoolSize, Integer.MAX_VALUE,
          DEFAULT_KEEPALIVE_MILLIS, MILLISECONDS,
          new DelayedWorkQueue());
}
```

```java
public class InBuiltThreadPool {
    public static void main(String[] args) {
        ExecutorService tp1 = Executors.newFixedThreadPool( nThreads: 3);  //core =3, max = 3
        ExecutorService tp2 = Executors.newSingleThreadExecutor(); // core = 1, max = 1
        ExecutorService tp3 = Executors.newCachedThreadPool(); // core = 0, max = Integer.MAX_VALUE;
        ExecutorService tp4 = Executors.newScheduledThreadPool( corePoolSize: 3); // core = 3, max = Integer.MAX_VALUE;
    }
}
```

OutOfMemoryError
Lock
synchronized
Lock interface
Synchronized
block
method
static method
class

```
class Demo {
public void method() {
synchronized(Demo.class) {
}
}
public synchronized void method() {

}
public synchronized static void method() {

}
public void method () {
synchronized(this) {
}
}
}
```

Lock interface

lock(), unlock(), newCondition(), tryLock(), lockInterruptibly()

ReentrantLock class

ReadWriteLock interface

method

Lock readLock();

Lock writeLock();

Class

reentrantReadWriteLock