Homework 5

* **Future / CompletableFuture**

**Future**

A Java Future represents the result of an asynchronous computation. When the asynchronous task is created, a Java Future object is returned. This Future object functions as a handle to the result of the asynchronous task. Once the asynchronous task completes, the result can be accessed via the future object returned when the task was started.

Java Future Interface:

public interface Future<V> {

boolean cancel(boolean mayInterruptIfRunning)

V get();

V get(long timeout, TimeUnit unit);

boolean isCancelled();

boolean isDone();

}

Get result from Future:

Future future = ... // get Future by starting async task

// do something else, until ready to check result via Future

// get result from Future

try {

Object result = future.get();

} catch (InterruptedException e) {

e.printStackTrace();

} catch (ExecutionException e) {

e.printStackTrace();

}

If you call the get() method before the asynchronous task has completed, the get() method will block until the result is ready.

try {

Object result =

future.get(1000, TimeUnit.MILLISECONDS);

} catch (InterruptedException e) {

} catch (ExecutionException e) {

} catch (TimeoutException e) {

// thrown if timeout time interval passes

// before a result is available.

}

This example waits for a maximum of 1000 milliseconds for the result to be available in the Future. If no result is available within 1000 milliseconds, a TimeoutException is thrown.

Cancel task via Future cancel()

future.cancel();

Check if task is done

Future future = ... // Get Future from somewhere

if(future.isDone()) {

Object result = future.get();

} else {

// do something else

}

Check if task is cancelled

Future future = ... // get Future from somewhere

if(future.isCancelled()) {

} else {

}

**CompletableFuture**

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and axtions that trigger upon its completion.

When two or more threads attempt to complete, completeExceptionally or cancel a Completable Future, only one of them succeeds.

CompletableFuture implements interface CompletionStage with the following policies:

* Actions supplied for dependent completions of non-async methods may be performed by the thread that completes the current CompletableFuture, or by any other caller of a completion method.
* All async method without an explicit Executor argument are performed using the ForkJoinPool.commonPool() (unless it does not support a parallelism level of at least two, in which case, a new Thread is created to run each task). To simplify monitoring, debugging, and tracking, all generated asynchronous tasks are instances of the marker interface CompletableFuture.AsynchronousCompletionTask.
* All CompletionStage methods are implemented independently of other public methods, so the bahavior of one method is not impact by override of others in subclasses.

CompletableFuture also implements Future with the following policies:

* Since (unlike FutureTask) this class has no direct control over the computation that causes it to be completed, cancellation is treated as just another form of exceptional completion. Method cancel has the same effect as completeExceptionally(new CancellationException()). Method isCompletedExceptionally() can be used to determine if a CompletableFuture completed in any exceptional fashion.
* In case of exceptional completion with a CompletionException, methods get() and get(long, TimeUnit) throw an ExecutionException with the same cause as held in the corresponding CompletionException. To simplify usage in most contexts, this class also defines methods join() and getNow(T) that instead throw the CompletionException directly in these cases.

1. **Summary of all the topics**

* Maven – Maven is a build automation tool used for Java projects.
  1. Type of repositories
     1. Local repository
        1. Use cd ~/.m2 can access
     2. Central repository – every developer can access
     3. Remote repository – belongs to company, need some credential (password etc.)
  2. Life cycle of Maven
     1. Validate
     2. Compile (all the source code in the project)
     3. Test
     4. Package (to jar/war file)
     5. Verify (run any checks to ensure the quality match, code coverage something)
     6. Install (install package into local repository)
     7. Deploy (to local or to company’s domain or to cloud service)
  3. Maven commend lines
     1. mvn clean
     2. mvn test
     3. mvn install
     4. (If you call one, the steps above in life cycle will be done as well)
  4. Create a project
     1. We put dependencies in pom.xml file

1. Git – tool used for version control
   1. Git vs svn – git has local and can work offline, svn must be used online
   2. We can create git in IntelliJ
   3. Commend lines
      1. Git branch
      2. Git add .
      3. Git commit -m “add read me file”
      4. Git push
   4. A token is needed to log in GitHub(?)
2. Eight basic date type
   1. Primitive type
      1. byte, short, int, long, float, double, char, boolean
   2. Wrapper class
      1. Byte, Short, Integer, Long, Float, Double, Character, Boolean
   3. Autoboxing and unboxing
      1. Autoboxing: primitive -> wrapper
      2. Unboxing: wrapper -> primitive
3. String/StringBuilder/StringBuffer
   1. String – immutable, thread safe
   2. StringBuilder – immutable, not thread safe
   3. StringBuffer – immutable, thread safe
      1. Slower than StringBuilder
4. String/String constant pool
   1. == compares reference address
   2. Object.equals() compares content (PS need to implement this method in our own classes, or default Object.equals() acts the same as ==
   3. Diagram

      Description automatically generated
   4. Integer constant pool has a range [-128, 127]
5. Equals / hashCode
6. Collection
   1. Diagram

      Description automatically generated
   2. List
      1. ArrayList – Resizable array, stored in consecutive space
         1. Get – O(1), support random access in ArrayList
         2. Remove – O(n)
      2. LinkedList – not stored in consecutive space
         1. Get – O(n)
         2. Remove – O(n)
   3. Vector – arraylist + synchronized (deprecated)
      1. Thread safe
   4. Stack
      1. Thread safe
      2. FILO
      3. Push, pop
   5. Deque: implementation is ArrayDeque
      1. first [ ] last
      2. replace stack: deque.offerFirst(), deque.pollFirst()
   6. Set
      1. HashSet – all element unique, unsorted, don’t keep the insertion order
      2. TreeSet – all element unique, sorted
      3. LinkedHashSet – all element unique, keep the insertion order
   7. Map
      1. HashMap – <key, value>
         1. Use key to get hashcode
         2. Hashcode % length
         3. Get index of the bucket
         4. Equals() to check
         5. Return value
         6. Table

            Description automatically generated
         7. If the LinkedList of an index has a length > 8, we transfer it into a red-black tree
      2. LinkedHashMap
      3. TreeMap – ordered by key
      4. HashTable – thread safe, has only one lock, only one thread can access at the same time
      5. ConcurrentHashMap – thread safe, 16 threads can access at the same time
   8. Queue
      1. FIFO
   9. Heap
      1. Use PriorityQueue to implement a heap
      2. minHeap – parent smaller that left and right, but don’t know whether left > right or right > left
      3. maxheap
   10. array
       1. int[] String[] Object[]
       2. int[][], char[][]
   11. Common interview questions
       1. list vs set
       2. HashMap vs HashTable vs ConcurrentHashMap
       3. HashSet implemented by a HashMap, just ignore value and keep the key
       4. TreeSet vs TreeMap
7. Comparator vs Comparable
   1. Code in IntelliJ
8. JVM
   1. Overview

Diagram

Description automatically generated

* 1. Class loader
  2. A picture containing diagram

     Description automatically generated
  3. Runtime data area

Diagram

Description automatically generated

* 1. Execution engine

Diagram, text

Description automatically generated

1. Class Loader
   1. Diagram

      Description automatically generated
2. Garbage Collection
   1. Serial GC – single thread
   2. Parallel GC – multi thread
   3. G1 GC – separate all the memory space into different chunks
      1. Table

         Description automatically generated
   4. CMS GC – concurrent mark and sweep GC
      1. Deprecated since java 9
      2. Completely removed in java 14
   5. Diagram

      Description automatically generated
      1. Minor GC
      2. Major GC
3. Keywords
   1. Keywords overview
      1. Reserved words

Diagram

Description automatically generated

* + 1. For data types (8)
       1. byte, short, int, long, float, double, char, boolean
    2. flow control (11)
       1. if, else, switch, case, default (2 kinds), for, do, while, break, continue, return
    3. modifiers (11)
       1. public, private, protected, static, final, abstract, synchronized, native, strictfp, transient, volatile
    4. exception handling (6)
       1. try, catch, finally, throw, throws, assert
    5. class related (6)
       1. class, package, import, extends, implements, interface
    6. object related (4)
       1. new, instanceof, super, this
  1. Final
     1. Final variable
        1. Create constant variable
        2. Must be initialized
     2. Final method
        1. Can’t be overridden
     3. Final class
        1. Can’t be extended
  2. Immutable class
     1. Final class
     2. Private final fields
     3. No setter
     4. Return deep copy of the collections for getter
  3. Static
     1. Block
     2. Variable
     3. Methods
     4. Classes
     5. Diagram

        Description automatically generated
  4. Implements vs extends
     1. Diagram

        Description automatically generated

1. OOP
   1. Abstraction – hide all the internal implementation
      1. Abstract class
      2. Interface
   2. Encapsulation
      1. Declare all the variables be private
      2. Provide getter and setter
   3. Inheritance
      1. Extends
      2. Implements
   4. Polymorphism
      1. Override
      2. Overloading
2. Exception
   1. Diagram

      Description automatically generated
   2. Checked exception vs unchecked exception
   3. How to handle the exception
      1. Try catch
      2. Throws
   4. How to customize exception
   5. How to handle multiple exceptions
      1. try {} catch (IOException ioe) {} catch (SQLException sqle) {} catch …
      2. try {} catch (IOException | SQL Exception | …) {}
      3. try { Connection con = DataDriver.getConnection();} catch (IOException ioe) {} catch (Exception ex) {} finally { if (con != null) {con.close();}}// from child to parent
3. Generics
   1. Easier and less error prone
   2. Enforce type correctness at compile time
   3. Without causing any extra overhead to your application
   4. Can check: <? Extends E>, <? Super T>, <T extends E>
4. IO stream
   1. Stream – a continuous flow of data
   2. Byte Stream – inherited from inputStream, outputStream
      1. Each time read 1 byte = 8 bits
   3. Character Stream – inherited from Reader, Writer
      1. Each time read 2 byte = 16 bits
   4. File – part of java.io
      1. Gives you access to underlying file systems
5. Serialization and deserialization
   1. Kafka – serialized (?)
6. Java 8 features
   1. Lambda
      1. (arguments) -> {body}
      2. Functional programming
      3. Less code
   2. Functional interface
      1. Predicate
         1. Public boolean test(T t)
      2. Function
         1. Public R apply(T t)
      3. Consumer
         1. Public void accept(T t)
      4. Supplier
         1. Public R get(T t)
   3. Optional
      1. used to prevent NPE
      2. If (obj == null) {

} else {

} // don’t have to use these

* 1. Stream API
     1. Intermediate operation: return a stream as result
        1. Map, flatmap, filter…
     2. Terminal operation: return nun-stream
        1. forEach, collect

1. Multi-threading
   1. Thread vs process
      1. Process – Independent memory space, heap, OS resources
      2. Thread – shared memory space, private stack, program counter, register
   2. Thread states
      1. New – thread create, not yet start
      2. Runnable – executing in JVM
      3. Blocked – wait for a monitor lock to enter synchronized block or method
      4. Waiting – object.wait with no timeout, thread.join() with no timeout, park()
      5. Timed waiting – thread sleep, Object.wait() with timeout, thread with timeout, park
      6. Terminated – thread has completed
      7. Diagram

         Description automatically generated
   3. Thread creation
      1. Extends thread
      2. Implements Runnable
      3. Implements callable
      4. Thread pool
   4. Runnable vs callable
      1. No return / has
      2. No exception / has
      3. Run() / call()
   5. Thread pool
   6. ThreadPoolExecutor
      1. corePoolSize
      2. maximumPoolSize
      3. keepAliveTime
      4. timeUnit
      5. workQueue
      6. threadFactory
      7. handler
         1. abortPolicy
         2. callerRunPolicy
         3. discardPolicy
         4. discardOldestPolicy
      8. goodnotes
   7. in-built thread pool
   8. OutOfMemoryError
   9. Lock
      1. synchronized
      2. Lock interface
   10. Synchronized
       1. Block
       2. Method
       3. Static method
       4. Class

Class demo {

Public void method() {

Synchronized(Demo.class) {

}

}

Public synchronized void method() {

}

Public synchronized static void method() {

}

Public void method() {

Synchronized(this) {

}

}

}

* 1. Lock interface
     1. Lock(), unlock(), newCondition(), tryLock(), lockInterruptibly()
     2. reentrantLock class
  2. ReadWriteLock interface
     1. method
        1. Lock readLock()
        2. Lock writeLock()
     2. Class
        1. reentrantReadWriteLock
  3. Future / CompletableFuture