1:

- checked vs unchecked exceptions
unchecked: error & runtime, everything else is checked
checked: must be handled, throw catch, try except;

- static vs non static methods
static: compile time, on ram, belongs to class, not instance of class, cannot override, can only access static methods
non static: runtime, belongs to object of class, can override, not on ram

- override vs overload
overload: same method, different parameters
override: same name, same parameters, one belongs to a parent class, one to a child class: provide specific implementation in the child for a method already implemented in parent

- interface vs abstract class
interface: all methods implicitly abstract, cannot have implementations, variables declared are final, can implement multiple interfaces, interface members public by default
abstract class: can have methods that implement a default behaviour, variables not final, can inherit only one class, can have private, protected members

- reference type vs primitive type
reference type: can be null, when compared with == the address is compared, not the value; created on the heap, where the garbage collector manages everything, are instantiable; classes in itself
primitive type: always has a value, take less memory, created on the stack

- process vs thread
process: execution unit where a program runs, has its own heap, stack, doesn’t share data; process management takes more system calls, takes more time to terminate
thread: execution unit, part of a process (in concurrent programming); share memory and data

- overriding mechanism\
how java achieves runtime polymorphism;
it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

3:

- thread

thread/runnable
execution unit (part of a process, more lightweight); usually operates on shared data and memory; less system calls to create/destroy;

- buffer, buffer operation, stream

buffer: portion of memory used to store streams of data coming from peripheral devices
stream: continuous flow of data, stored in contiguous memory locations in the buffer; intermediate to main memory and cpu

- garbage collector

daemon thread always running in the background; disposes of unused objects (memory); unreachable objects are deallocated

- multithreading

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.
Threads can be created by using two mechanisms :
1. Extending the Thread class
2. Implementing the Runnable Interface

- synchronization for threads

can use synchronized blocks in java; synchronized methods; shared variables; mutexes; semaphore; barrier; conditional variables;

- possible states of a thread

new (created, ready to be run) not alive
runnable, .start(), passed to scheduler that decides when/what happens with the thread
blocked (when thread cannot execute next step of operation immediately)
waiting, .wait()
timed waiting, sleep(miliseconds)
terminated – completed task, forcefully killed or error

- monitor model

every java obj has a monitor (abstract data type); has private variables and methods; has the property that operations are synchronized: a monitor operation can be accessed by a single thread at a time

- event driven programming

programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs or threads. Event-driven programming is the dominant paradigm used in graphical user interfaces

- executor service
framework for asynchronous execution; contains a pool of threads; tasks come in, threads who are free execute those tasks

- blocking queue
queue with no concurrency issues: multiple threads can add and remove elements from the queue; if it reaches upper bound, will be blocked from producer until a consumer takes element/s out

- concurrent collections
contain methods through which we can get a synchronized version of non-synchronized objects; don't have to take care of thread safety; can use multiple threads on object

- forkjoinpool
attempt to speed up parallel processing by using all the cores; implementation of executor service that manages the worker; divide et impera; each thread has a queue; free threads to "steal" from queues of busy threads; new task created doesn't mean a new thread is created; double ended queue => deque

- countdownlatch
can block a thread until other threads have completed a given task
wait for all threads to count down to 0; counter cannot be reset

- cyclic barrier
synchronizer that allows a set of threads to wait for each other once the execution has reached a certain point; takes a single integer that denotes the number of threads that need to call the *await()* method on the barrier instance to signify reaching the common execution point:

- semaphore
controls access to a shared variable via a counter: if the counter has a value grater than 1, access is permitted, otherwise denied; thread done: releases permit, counter increased, other threads can gain access to the resource

- atomic variable
when an operation is started on an atomic variable, it will surely finish without any other treads doing anything on the variable; atomic operations – eg, no need for lock if we want to increment an atomic variable in a thread; better performance than locks