Concurrency 1:

Aka parallelism

* Useful for
  + Allow programs to interact with other systems/users
  + Multi-core systems

When multiple processes perform at the same time include any interaction between them

* Each process allocated certain # of resources and run in parallel
* E.g. having multiple windows open
* If one core allocates separate time slices to different processes, can give the illusion of concurrency but it isn’t
* Usually a combination of using multiple cores and time slicing

Threads

* Treated as a lightweight process
* Threads exist within a process, can have multiple within a process
* Threads can share memory, so no memory has to be allocated each time

Multi-threading

* OS can run threads in parallel
* the threads share a local memory but have their own local variables
* Virtually can be done on a single processor, actually done on a multi-core processor

Issues

* Nondeterminism
  + The order of tasks may not always be 100% clear
  + This can lead to pipelining issues, e.g. if task 3 needs input of task 2
  + Or if one task takes longer than the other but both need to be completed for going on then time is wasted waiting for task completion
  + Do not know which thread will run 1st or 2nd, because OS schedules the threads
* Liveness
  + Normally just care whether the code works correctly
  + Now need to worry that code performs correctly in time
  + Good things happen eventually (give results in an acceptable time)
  + Need to determine what level of eventually is acceptable
    - Depends on the application

Multi-tasking

* Multiple processes
  + Threads are within a process
* Each has their own memory space
* Can run on a single or multi core processor
  + Depends on algorithm

How to create threads

* Extend the Thread class
* E.g. public class HelloThread extends Thread {}
* Easier to implement
* However cannot extend the class that extends Thread
* Inherit a lot of unnecessary things from Thread class
* Using Runnable interface
* E.g. public class HelloRunnable implements Runnable {}
* Stopping a thread
  + Set a flag
  + Loop will exit and thread will stop on next iteration after done is set to true
  + Public void run () {
    - While (!done) {
      * //..
    - }
  + }
  + Interrupting
  + When isInterrupted is called, the thread is stopped

Race Conditions

* Data synchronisation between threads can be an issue
* 2 threads try to access or change data at the same time

Synchronisation

* Use synchonrize(d) keyword in methods in the class
* When 1 thread executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block suspend execution until the 1st thread is done with the object

Can synchronise a set of statements, not necessarily a whole method

e.g. synchronised(this){

..//everything In here is synchronised

}

//everything out here is not synchronised

Check out atomics