Some multithreaded applications would be:

1. Web Browsers - A web browser can download any number of files and web pages (multiple tabs) at the same time and still lets you continue browsing. If a particular web page cannot be downloaded, that is not going to stop the web browser from downloading other web pages.
2. Web Servers - A threaded web server handles each request with a new thread. There is a thread pool and every time a new request comes in, it is assigned to a thread from the thread pool.
3. Computer Games - You have various objects like cars, humans, birds which are implemented as separate threads. Also playing the background music at the same time as playing the game is an example of multithreading.
4. Text Editors - When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads. The same applies for Word processors also.
5. IDE - IDEs like Android Studio run multiple threads at the same time. You can open multiple programs at the same time. It also gives suggestions on the completion of a command which is a separate thread.

I use multithreading mostly in web applications - Typically in the DAO layer, the database (JDBC or Hibernate/ORM) code loads all records that were requested by the user, page size irrelevant. If the user wants 50,000 rows then so be it - but that takes a while to retrieve and on which to work.

So, I would usually have another object that is implemented Runnable or extend Thread (but usually implementing Runnable) do the loading and it would have a variable with public getter (and maybe setter). If implementing Runnable as I usually do, I’d probably also extend Observable and whenever a new record was added to the variable (list/ map /set / whatever), it would notify observers and when X rows were accumulated, the servlet will return to the front end Javascript with the results. Subsequently when the user clicks “Next Page” or to click on another page altogerther, those parameters would be passed back to the servlet and from the collection of rows available, that subset would be retrieved from the larger set and returned (probably in JSON format). By that time though, usually the entire recordset would have been loaded and available to pull from then and one wouldn’t have to wait for the database code to load and create objects for each row up until where the page started. When the operation was done then, the thread would stop - the object still exists with the collection inside but the thread was no longer doing its thing so you can retrieve the collection or continue using it out of the object.

**1)File processing:** In file processing, you can divide data in chunks and get it processed in multiple threads.

**2)GUI:**GUI events may have some time consuming tasks in main thread. It is advisable to exploit the power of parallel processing by dividing the task into smaller parts.

**3) Rendering Images.**

**4) Brute-Force Calculations.**

(Basically tasks that use repeated operations to reach results and may or may not be mutually exclusive.)

Hope this was useful.!!

<https://www.journaldev.com/1079/multithreading-in-java>