CHANDAN MUKHERJEE

MTech (IT), BE (Computer Science)
SCJP (Java), Oracle (SQL) GLOBAL CERTIFIED
Microsoft Certified Innovative Educator
Corporate Trainer
chan.muk@gmail.com

THREAD / MULTITHREAD

- ✓ THREAD HELPS US TO DO MULTITASKING.
- ✓ THREAD HELPS US TO EXECUTE OUR CODE.
- ✓ THREAD IS AN OBJECT OF SYSTEM DEFINED THREAD CLASS.

THREAD LIFE CYCLE

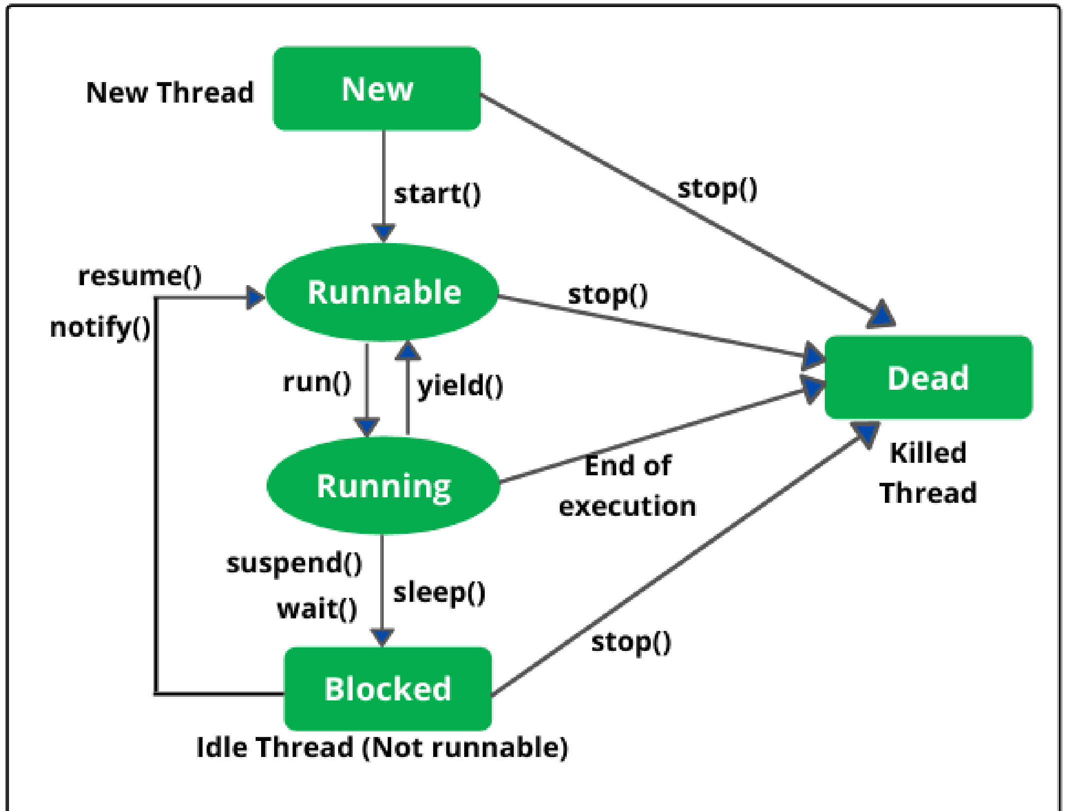
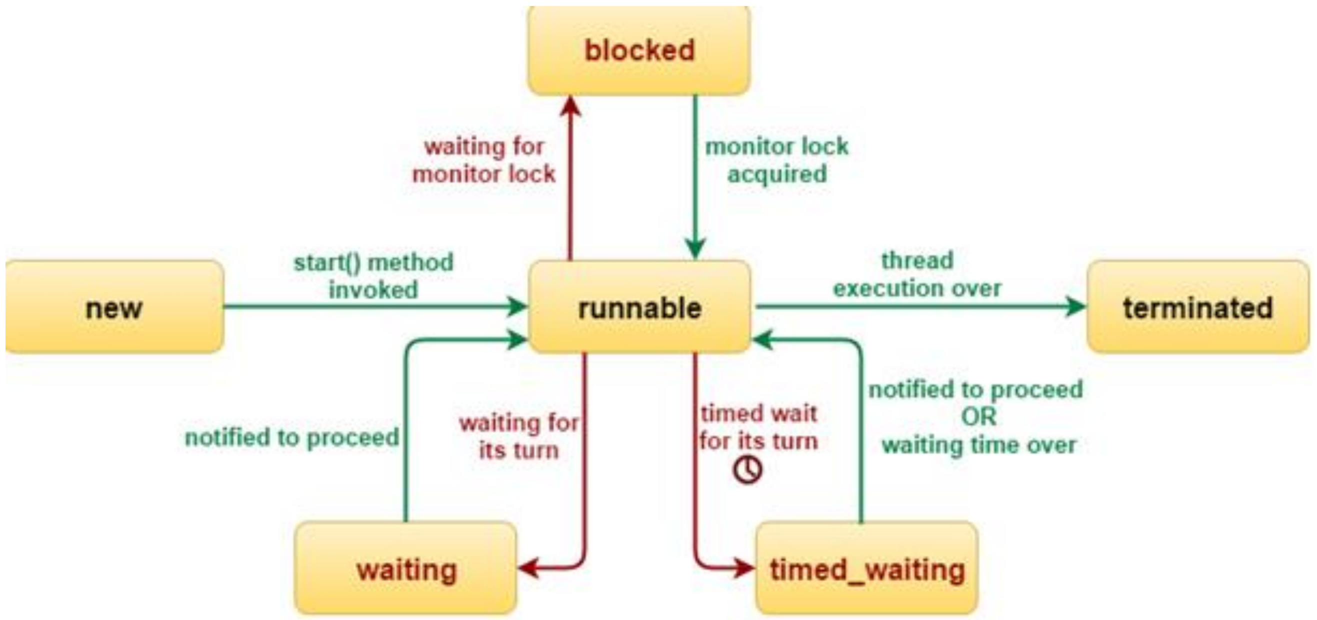


Fig: State Transition Diagram of a Thread



How to create Thread in Java

Two ways to create thread in Java

- By extending Thread class
- By implementing Runnable Interface

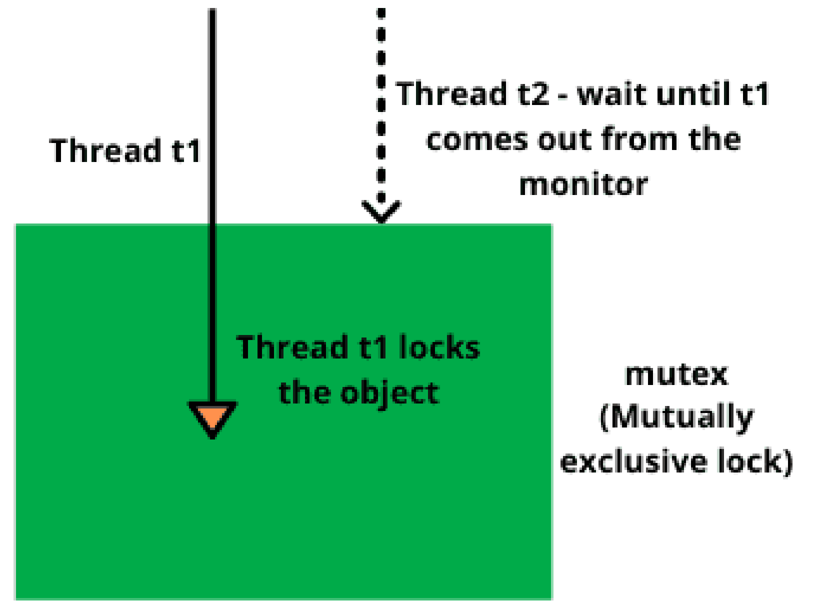

Thread t1
Thread t2 - wait until t1 comes out from the monitor
Thread t1 locks the object
mutex (Mutually exclusive lock)

Fig: Thread Synchronization in Java

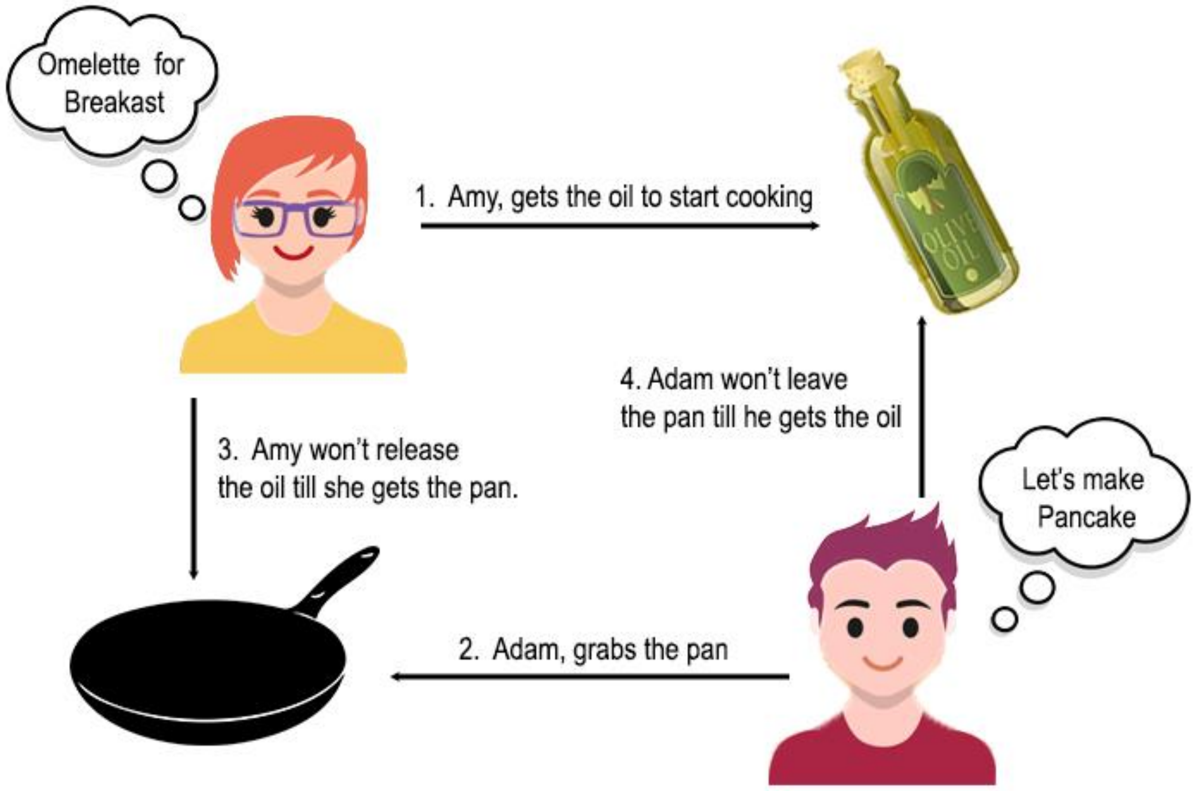
wait(), notify(), and notifyAll()

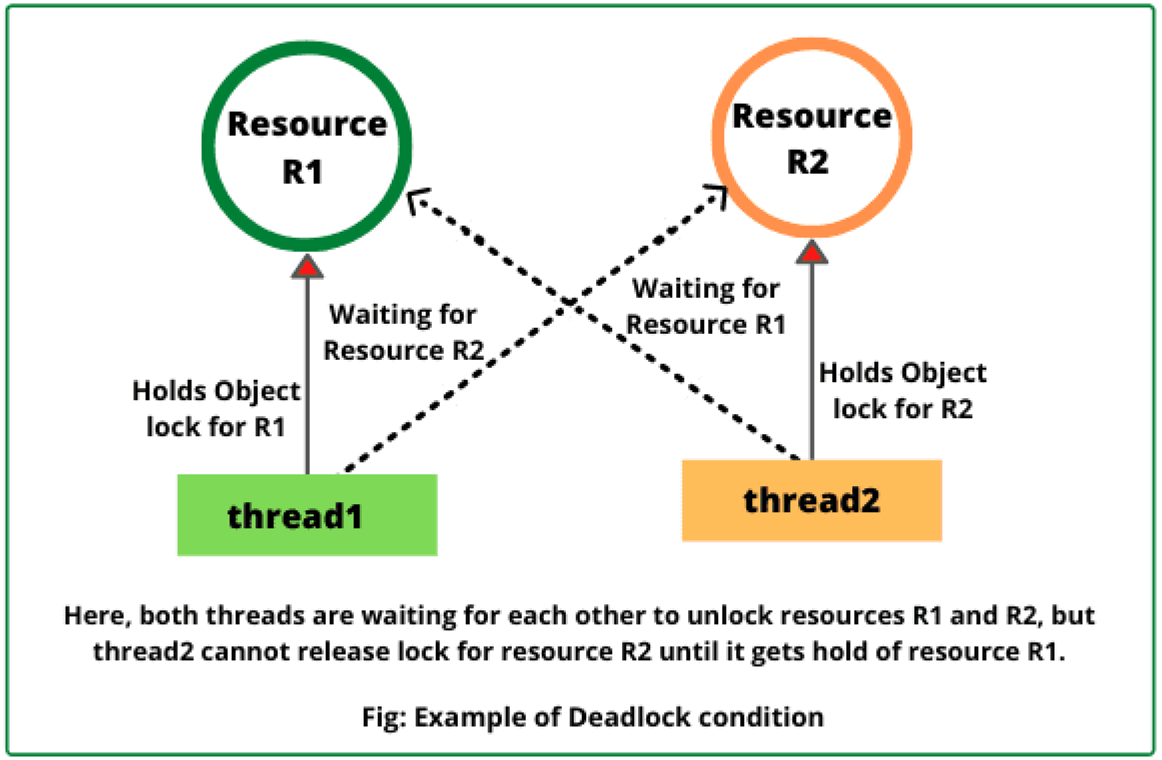
Use the wait(), notify(), and notifyAll() methods to facilitate communication among threads.

The wait(), notify(), and notifyAll() methods must be called in a synchronized method or a synchronized block on the calling object of these methods. Otherwise, an IllegalMonitorStateException would occur.

The wait() method lets the thread wait until some condition occurs. When it occurs, you can use the notify() or notifyAll() methods to notify the waiting threads to resume normal execution. The notifyAll() method wakes up all waiting threads, while notify() picks up only one thread from a waiting queue.

DEADLOCK


Omelette for Breakast
1. Amy, gets the oil to start cooking
2. Adam, grabs the pan
3. Amy won't release the oil till she gets the pan.
4. Adam won't leave the pan till he gets the oil
Let's make Pancake
OLIVE OIL



Here, both threads are waiting for each other to unlock resources R1 and R2, but thread2 cannot release lock for resource R2 until it gets hold of resource R1.

Fig: Example of Deadlock condition

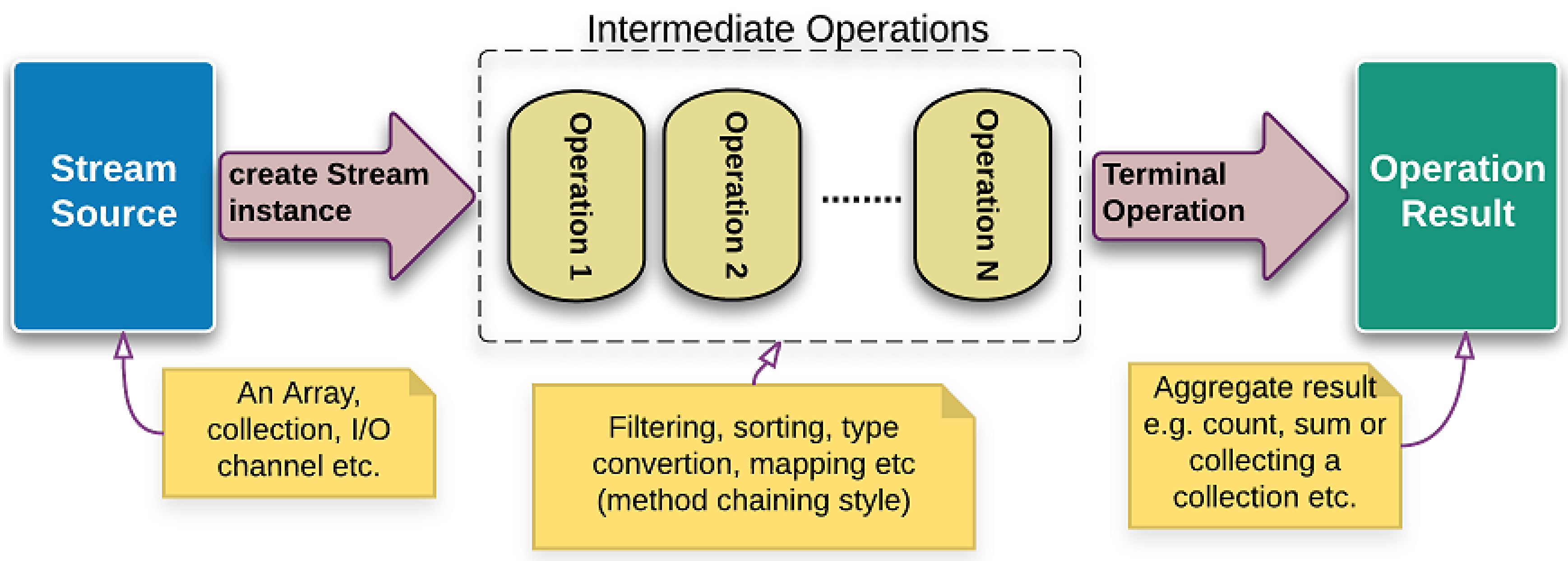
STREAM

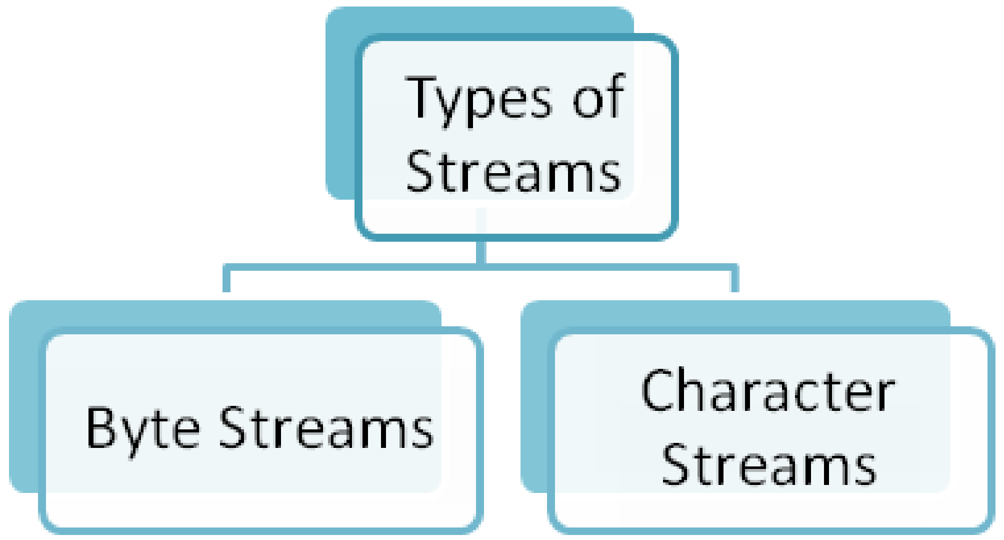
A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

- A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined.
- Terminal operations mark the end of the stream and return the result.

Java Streams


Intermediate Operations
Stream Source
create Stream instance
Operation 1
Operation 2
....
Operation N
Terminal Operation
Operation Result
An Array, collection, I/O channel etc.
Filtering, sorting, type convertion, mapping etc (method chaining style)
Aggregate result e.g. count, sum or collecting a collection etc.



Character streams	Byte streams
Meant for reading or writing to character- or text-based I/O such as text files, text documents, XML, and HTML files.	Meant for reading or writing to binary data I/O such as executable files, image files, and files in low-level file formats such as `.zip`, `.class`, `.obj`, and `.exe`.
Data dealt with is 16-bit Unicode characters.	Data dealt with is bytes (i.e., units of 8-bit data).
Input and output character streams are called *readers* and *writers*, respectively.	Input and output byte streams are simply called *input streams* and *output streams*, respectively.
The abstract classes of `Reader` and `Writer` and their derived classes in the `java.io` package provide support for character streams.	The abstract classes of `InputStream` and `OutputStream` and their derived classes in the `java.io` package provide support for byte streams.

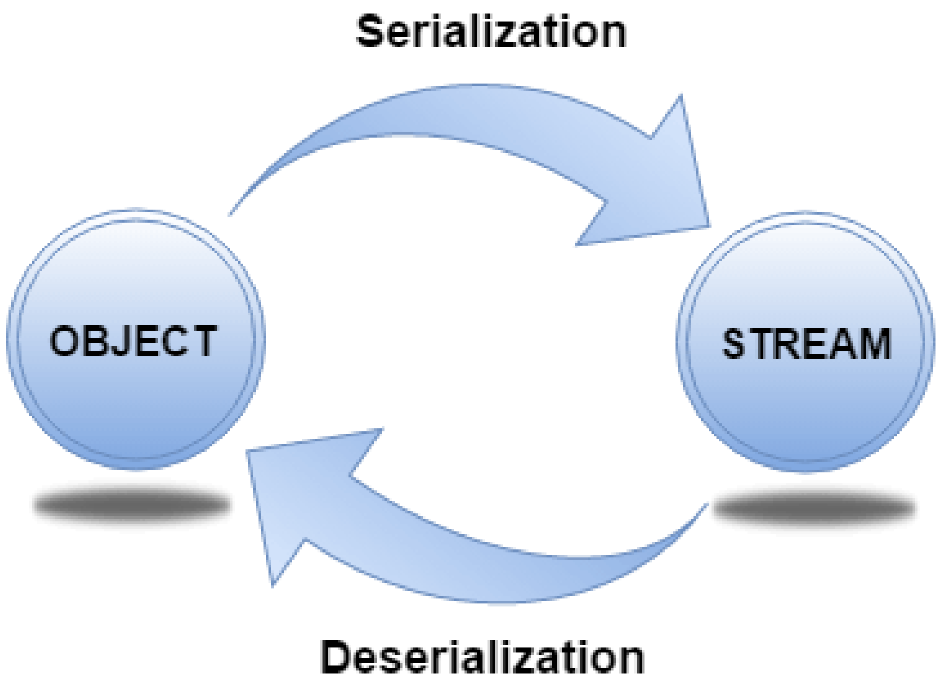
Character Stream
16 bits carrier - Unicode

- **Reader**
  - BufferedReader
    Used for Buffered Input Stream
  - CharArrayReader
    Used for reading from an array
  - StringReader
    Used for read from a string
  - FileReader - Used for reading from a File
  - PipedReader - Input pipe
  - InputStreamReader - translates bytes to chatacter
  - FilterReader - filtered reader
  - LineNumberReader - used to count lines
- **Writer**
  - BufferedWriter
    Used for Buffered Output Stream
  - CharArrayWriter
    Used for writing into an array
  - StringWriter
    Used for write into a string
  - FileWriter - Used for writing into a File
  - PipedWriter - Output pipe
  - OutputStreamWriter - characters to bytes
  - FilterWriter - filtered writer
  - PrintStream - Contains **print( )** and **println( )**

Byte Stream

8 bits carrier

- InputStream
  - BufferedInputStream
    Used for Buffered Input Stream
  - ByteArrayInputStream
    Used for reading from a byte array
  - DataInputStream
    Used for reading java standard data type
  - ObjectInputStream - Input stream for objects
  - FileInputStream - Used for reading from a File
  - PipedInputStream - Input pipe
  - InputStream - Describe stream input
  - FilterInputStream - Implements **InputStream**
- OutputStream
  - BufferedOutputStream
    Used for Buffered Output Stream
  - ByteArrayOutputStream
    Used for writing into a byte array
  - DataOutputStream
    Used for writing java standard data type
  - ObjectOutputStream- Output stream for objects
  - FileOutputStream- Used for writing into a File
  - PipedOutputStream - Output pipe
  - OutputStream- Describe stream output
  - FilterOutputStream- Implements **OutputStream**
  - PrintStream - Contains **print( )** and **println( )**

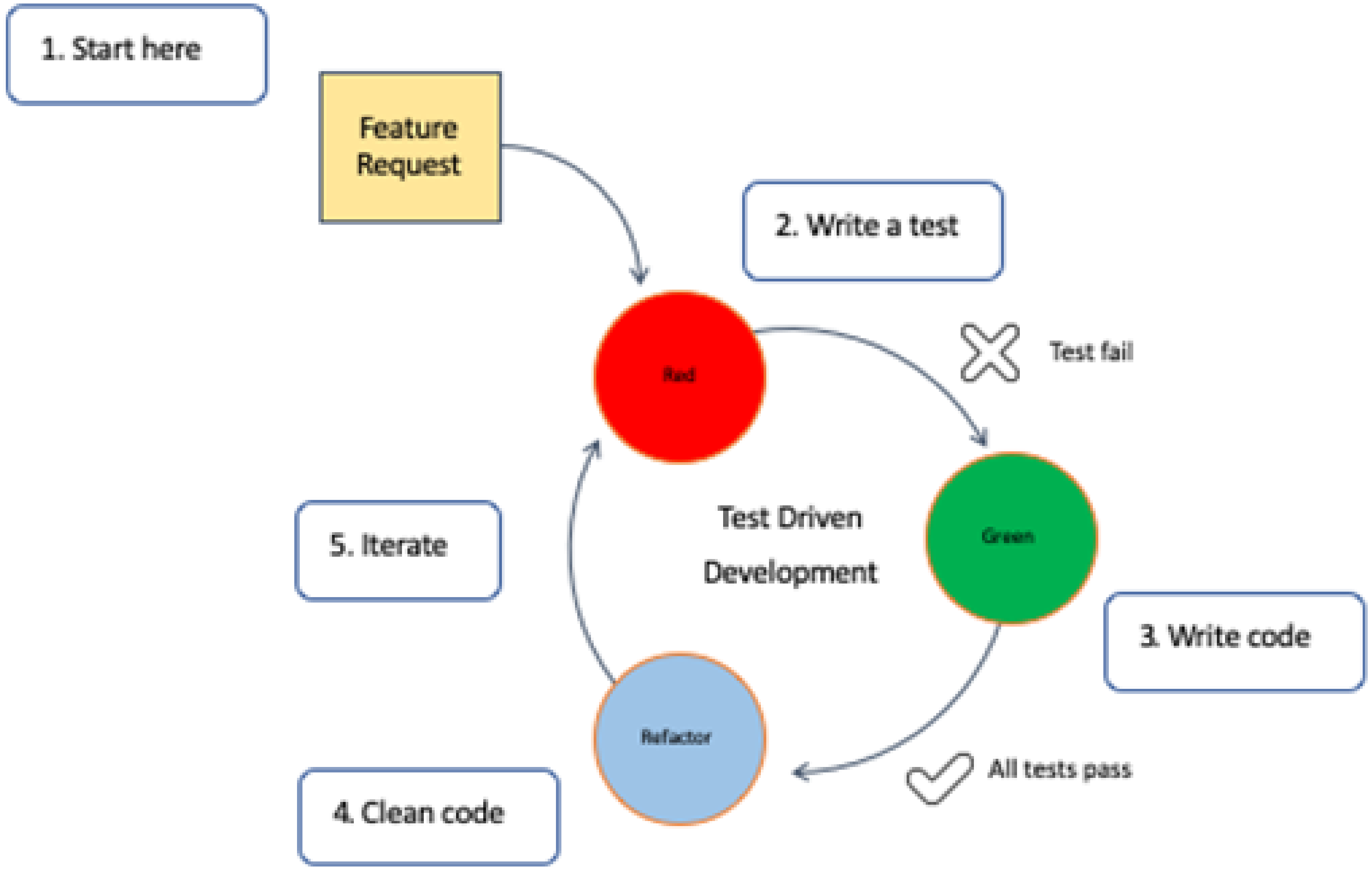

Serialization
OBJECT
STREAM
Deserialization

TDD – Test Driven Development

- Iterative development process.
- Every iteration starts with a set of tests written for a new piece of functionality.
- Test cases are created before code is written
- TDD instructs developers to write new code only if an automated test has failed.

- Five steps of test-driven development

1. Read, understand, and process the feature or bug request.
2. Translate the requirement by writing a unit test. If you have hot reloading set up, the unit test will run and fail as no code is implemented yet.
3. Write and implement the code that fulfills the requirement. Run all tests and they should pass, if not repeat this step.
4. Clean up your code by refactoring.
5. Rinse, lather and repeat.



:atus of the

l way.
s covered with

https://developer.ibm.com/articles/5-steps-of-test-driven-development/