**FUNCTIONAL PROGRAMMING**

Functional Programming

Functions are First class Values (as equal as int or String).

Functions like other languages.

Functions as variables/objects.

Functions as argument or return value of function.

Functions defined in other function.

Functions as anonymous.

Output of functions are mapped to their inputs.  
Immutability – produce new result (instead of changing in place)  
Function result depends solely on the input and also doesn't modify state of args or other objects.

Functions should be referentially transparent i.e. it should be easily replaceable by its result without changing program semantic.

Such functions are said to be side-effect free functions or pure functions.

**Stream Introduction**  
Java 8 Stream is NOT IO streams.  
Java streams represents pipeline of operations through which data is processed.  
Stream operations are of two types  
Intermediate operations: Yields another stream.  
filter()  
map(), flatMap()  
limit(), skip()  
sorted(), distinct()  
Terminal operations: Yields some result.  
reduce()  
forEach()  
collect(), toArray()  
count(), max(), min()  
Stream operations are higher order functions (take functional interfaces as arg).

Java stream characteristics  
No storage: Stream is an abstraction. Stream doesn’t store the data elements. They are stored in source collection or produced at runtime.  
Immutable: Any operation doesn’t change the stream itself. The operations produce new stream of results.  
Lazy evaluation: Stream is evaluated only if they have terminal operation. If terminal operation is not given, stream is not processed.  
Not reusable: Streams processed once (terminal operation) cannot be processed again

Serialization

Converting state of object into a sequence of bytes is referred as Serialization. The sequence of bytes includes object data as well as metadata.

Serialized data can be further saved into a file (using FileOutputStream) or sent over the network (Marshalling process).

Converting (serialized) bytes back to the Java object is referred as Deserialization.

These bytes may be received from the file (using FileInputStream) or from the network (Unmarshalling process).

Serializable interface

Object can be serialized only if class is inherited from Serializable interface; otherwise writeObject() throws NotSerializableException.  
Serializable is a marker interface.

Each serializable class is associated with a version number, called a serialVersionUID.  
It is recommended that programmer should define it as a static final long field (with any access specifier). Any change in class fields expected to modify this  
serialVersionUID.

private static final long serialVersionUID = 1001L;

During deserialization, this number is verified by the runtime to check if right version of the class is loaded in the JVM. If this number mismatched, then  
InvalidClassException will be thrown.

Program  
Program is set of instructions given to the computer.  
Executable file is a program.  
Executable file contains text, data, rodata, symbol table, exe header.  
Process  
Process is program in execution.  
Program (executable file) is loaded in RAM (from disk) for execution. Also OS keep information required for execution of the program in a struct called PCB  
(Process Control Block).  
Process contains text, data, rodata, stack, and heap section.  
Thread  
Threads are used to do multiple tasks concurrently within a single process.  
Thread is a lightweight process.  
When a new thread is created, a new TCB is created along with a new stack. Remaining sections are shared with parent process.  
Process vs Thread  
Process is a container that holds resources required for execution and thread is unit of execution/scheduling.  
Each process have one thread created by default -- called as main thread.

Multi-threading (Java)  
Java applications are always multi-threaded.  
When any java application is executed, JVM creates (at least) two threads.  
main thread -- executes the application main()  
GC thread -- does garbage collection (release unreferenced objects)  
Programmer may create additional threads, if required.

Java doesn't support multiple inheritance. If your class is already inherited from a super class, you cannot extend it from Thread class. Prefer Runnable in this  
case; otherwise you may choose any method.

**Daemon threads**  
By default all threads are non-daemon threads (including main thread).  
We can make a thread as daemon by calling its setDaemon(true) method -- before starting the thread.  
Daemon threads are also called as background threads and they support/help the non-daemon threads.  
When all non-daemon threads are terminated, the Daemon threads get automatically terminated.

**Synchronization**

When multiple threads try to access same resource at the same time, it is called as Race condition.  
Example: Same bank account undergo deposit() and withdraw() operations simultaneously.  
It may yield in unexpected/undesired results.  
This problem can be solved by Synchronization.  
The synchronized keyword in Java provides thread-safe access.  
Java synchronization internally use the Monitor object associated with any object. It provides lock/unlock mechanism.  
"synchronized" can be used for block or method.  
It acquires lock on associated object at the start of block/method and release at the end. If lock is already acquired by other thread, the current thread is  
blocked (until lock is released by the locking thread).  
"synchronized" non-static method acquires lock on the current object i.e. "this". Example: