**SERIALIZATION**

You can think of serialization as the process of converting an object instance into a sequence of bytes (which may be binary or not depending on the implementation).

It is very useful when you want to transmit one object data across the network, for instance from one JVM to another.

In Java, the serialization mechanism is built into the platform, but you need to implement the Serializable interface to make an object serializable.

You can also prevent some data in your object from being serialized by marking the attribute as transient.

The **ObjectOutputStream** class is used to write primitive data types and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

An **ObjectInputStream** deserializes objects and primitive data written using an ObjectOutputStream.

**Example of Java Serialization**

In this example, we are going to serialize the object of Student class. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

import java.io.\*;

class Persist{

public static void main(String args[])throws Exception{

Student s1 =new Student(211,"ravi");

FileOutputStream fout=new FileOutputStream("f.txt");

ObjectOutputStream out=new ObjectOutputStream(fout);

out.writeObject(s1);

out.flush();

System.out.println("success");

}

}

**Example of Java Deserialization**

import java.io.\*;

class Depersist{

public static void main(String args[])throws Exception{

ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));

Student s=(Student)in.readObject();

System.out.println(s.id+" "+s.name);

in.close();

}

}

**THREADING**

A thread is a:

* Facility to allow multiple activities within a single process
* A thread is a series of executed statements
* Each thread has its own program counter, stack and local variables
* A thread is a nested sequence of method calls

**What’s the need of a thread or why we use Threads?**

* To perform asynchronous or background processing
* Increases the responsiveness of GUI applications
* Take advantage of multiprocessor systems
* Simplify program logic when there are multiple independent entities

**What happens when a thread is invoked?**

When a thread is invoked, there will be two paths of execution. One path will execute the thread and the other path will follow the statement after the thread invocation. There will be a separate stack and memory space for each thread.

**Risk Factor**

* Proper co-ordination is required between threads accessing common variables [use of synchronized and volatile] for consistence view of data
* overuse of java threads can be hazardous to program’s performance and its maintainability.

Simply put, a thread is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time. Threads allows us to do this.

**Multithreaded** applications deliver their potent power by running many threads concurrently within a single program. From a logical point of view, multithreading means multiple lines of a single program can be executed at the same time, however, it is not the same as starting a program twice and saying that there are multiple lines of a program being executed at the same time. In this case, the operating system is treating the programs as two separate and distinct processes.

**public void run():** is used to perform action for a thread.

**public void start():** starts the execution of the thread.JVM calls the run() method on the thread.

**public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

**public boolean isAlive():** tests if the thread is alive

**public void stop():** is used to stop the thread(depricated).

**Thread creation in Java**

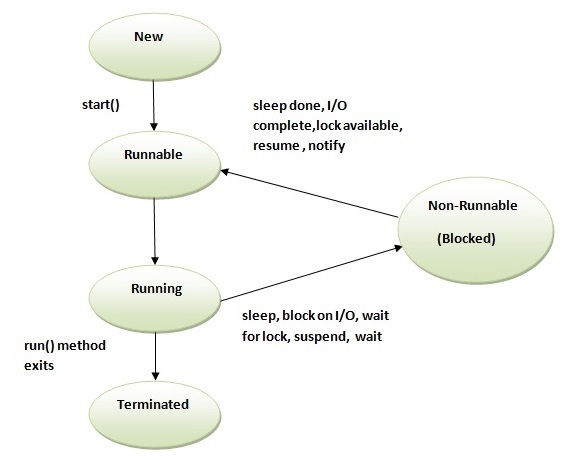
Thread implementation in java can be achieved in two ways:

1. Extending the java.lang.Thread class
2. Implementing the java.lang.Runnable Interface

**Extends Thread class vs Implements Runnable Interface?**

* Extending the Thread class will make your class unable to extend other classes, because of the single inheritance feature in  JAVA. However, this will give you a simpler code structure. If you implement Runnable, you can gain better object-oriented design and consistency and also avoid the single inheritance problems.
* If you just want to achieve basic functionality of a thread you can simply implement Runnable interface and override run() method. But if you want to do something serious with thread object as it has other methods like suspend(), resume(), ..etc which are not available in Runnable interface then you may prefer to extend the Thread class.

**Life cycle of a Thread (Thread States)**



**1) New** - The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

**2) Runnable** - The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running** - The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked**) - This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated** - A thread is in terminated or dead state when its run() method exits.

**GARBAGE COLLECTION**

In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

Advantage of Garbage Collection

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

**How can an object be unreferenced?**

1) By nulling a reference:

Employee e=**new** Employee();

e=**null**;

2) By assigning a reference to another:

Employee e1=**new** Employee();

Employee e2=**new** Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection

3) By annonymous object:

**new** Employee();

**finalize() method**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

**gc() method**

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

The garbage collector is a program which runs on the Java Virtual Machine which gets rid of objects which are not being used by a Java application anymore. It is a form of automatic memory management.

When a typical Java application is running, it is creating new objects, such as Strings and Files, but after a certain time, those objects are not used anymore. For example, take a look at the following code:

for (File f : files) {

String s = f.getName();

}

In the above code, the String s is being created on each iteration of the for loop. This means that in every iteration, a little bit of memory is being allocated to make a String object.

Going back to the code, we can see that once a single iteration is executed, in the next iteration, the String object that was created in the previous iteration is not being used anymore -- that object is now considered "garbage".

Eventually, we'll start getting a lot of garbage, and memory will be used for objects which aren't being used anymore. If this keeps going on, eventually the Java Virtual Machine will run out of space to make new objects.

That's where the garbage collector steps in.

The garbage collector will look for objects which aren't being used anymore, and gets rid of them, freeing up the memory so other new objects can use that piece of memory.

**GUI (Swing, AWT, APPLETS)**

**Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

**Java Swing** tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components. It guarantees that your user interface design will look the same on different platforms.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

**Applet** is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

o It works at client side so less response time.

o Secured

o It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

o Plugin is required at client browser to execute applet.

**FILE HANDLING**

File Handling in java comes under IO operations. Java IO package java.io classes are specially provided for file handling in java. File Handling in java is done by using various input and output streams

Some of the common file handling operations are;

**Create File**

We can use File class createNewFile() method to create new file. This method returns true if file is successfully created, otherwise it returns false. Below is a simple program showing how to create a new file in java.

File file = new File("data.txt");

try {

boolean createNewFile = file.createNewFile();

} catch (IOException e) {

// TODO Auto-generated catch block

}

**Delete File**

File class delete method is used to delete a file or an empty directory. Below is a simple example to delete a file.

File file = new File("data.txt");

boolean delete = file.delete();

**Read File**

There are many ways to read a file in java. We can use BufferedReader, FileReader or Files class. Below code snippet is to read file line by line.

File file = new File("data.txt");

FileInputStream fis = new FileInputStream(file);

InputStreamReader isr = new InputStreamReader(fis, cs);

BufferedReader br = new BufferedReader(isr);

String line;

while((line = br.readLine()) != null){

System.out.println(line);

}

br.close();

**Write File**

We can use FileWriter, BufferedWriter, Files or FileOutputStream to write file in java. Below code snippet use Stream to write data to file.

OutputStream os = null;

try {

os = new FileOutputStream(new File("/Users/pankaj/os.txt"));

os.write(data.getBytes(), 0, data.length());

} catch (IOException e) {

e.printStackTrace();

}finally{

try {

os.close();

} catch (IOException e) {

e.printStackTrace();

}

}

**JDBC**

**ORM**

MyBatis is SQL centric. It heps you calling SQL statements and mapping results (tables) to object trees.

The main benefit is that it is not an ORM. It does not map tables to object so does not suffer the orm impedance mismatch. Fits well for complex or legacy databases or to use db features like stored procedures, views and so.

**TRADITIONAL METHODOLGY**

**Waterfall**

Often considered the classic approach to the systems development life cycle, the waterfall model describes a development method that is linear and sequential.

Once a phase of development is completed, the development proceeds to the next phase and there is no turning back.

**Advantages of the Waterfall Methodology**

1. The waterfall methodology stresses meticulous record keeping. Having such records allows for the ability to improve upon the existing program in the future.

2. With the waterfall methodology, the client knows what to expect. They’ll have an idea of the size, cost, and timeline for the project. They’ll have a definite idea of what their program will do in the end.

3. In the case of employee turnover, waterfall’s strong documentation allows for minimal project impact.

**Disadvantages of the Waterfall Methodology**

1. Once a step has been completed, developers can’t go back to a previous stage and make changes.

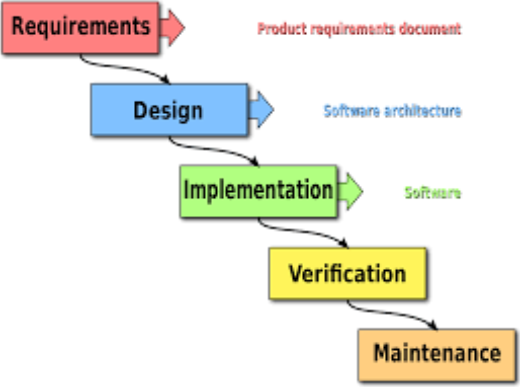
2. Waterfall methodology relies heavily on initial requirements. However, if these requirements are faulty in any manner, the project is doomed.

3. If a requirement error is found, or a change needs to be made, the project has to start from the beginning with all new code.

4. The whole product is only tested at the end. If bugs are written early, but discovered late, their existence may have affected how other code was written.

Additionally, the temptation to delay thorough testing is often very high, as these delays allow short-term wins of staying on-schedule.

5. The plan doesn’t take into account a client’s evolving needs. If the client realizes that they need more than they initially thought, and demand change, the project will come in late and impact budget.



**Iterative**

In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

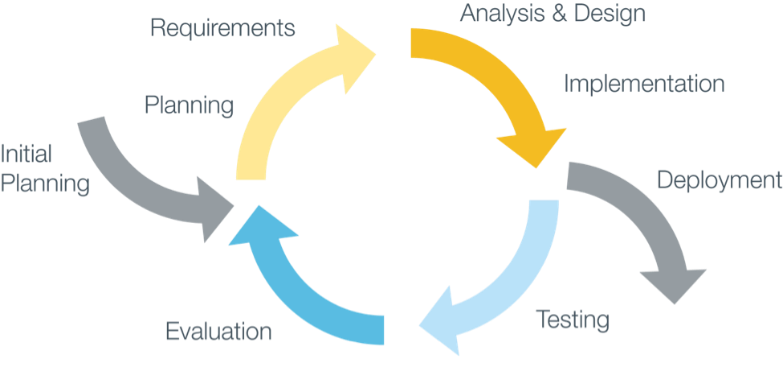
An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

**Advantages of Iterative model:**

* In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
* In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
* In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
* In iterative model less time is spent on documenting and more time is given for designing.

**Disadvantages of Iterative model:**

* Each phase of an iteration is rigid with no overlaps
* Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle



**AGILE METHODOLOGY**

Agile software development is a group of software development methods based on iterative and incremental development.

It uses where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change.

It is a conceptual framework that promotes interactions throughout the development cycle.

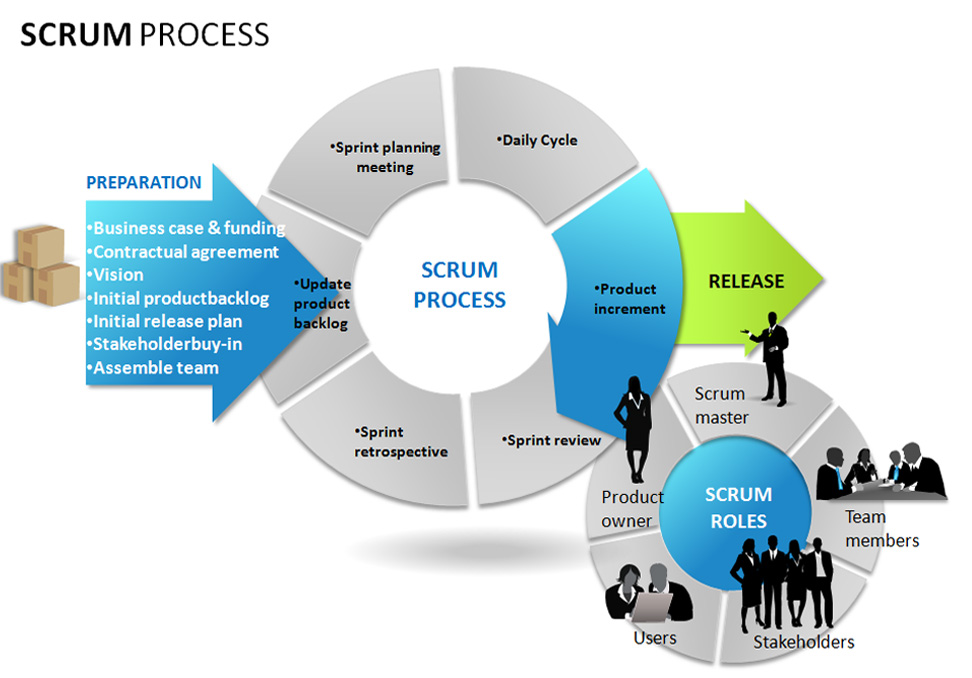
**Scrum**

Scrum is a simple and repeatable way of managing work. It can be used for projects, or for ongoing activities. It was originally designed for software development work, although is not specific to software development so can be used to manage any work.

With scrum, the product is built in a series of fixed-length iterations called sprints that give teams a framework for shipping software on a regular cadence.

Scrum calls for four [ceremonies](https://www.atlassian.com/agile/ceremonies) that bring structure to each sprint:

* **Sprint planning:** A team planning meeting that determines what to complete in the coming sprint.
* **Daily stand-up:** Also known as a daily scrum, a 15-minute mini-meeting for the software team to sync.
* **Sprint demo:** A sharing meeting where the team shows what they've shipped in that sprint.
* **Sprint retrospective:** A review of what did and didn't go well with actions to make the next sprint better.



With Scrum methodology, the “Product Owner” works closely with the team to identify and prioritize system functionality in form of a “Product Backlog”. The Product Backlog consists of features, bug fixes, non-functional requirements, etc. – whatever needs to be done in order to successfully deliver a working software system. With priorities driven by the Product Owner, cross-functional teams estimate and sign-up to deliver “potentially shippable increments” of software during successive Sprints, typically lasting 30 days. Once a Sprint’s Product Backlog is committed, no additional functionality can be added to the Sprint except by the team. Once a Sprint has been delivered, the Product Backlog is analyzed and reprioritized, if necessary, and the next set of functionality is selected for the next Sprint.



