Java

Date / /
Page L&W

* Defination of JVM Architecture
- JVM is a part of Java Runtime Environment that executes Java bytecode. It provides platform independence by allowing the same java program to run on any os.

• key concepts.

(a) Class Loader :- Loads compiled .class files (bytecode) into JVM memory.
(b) Bytecode Verifier :- Ensures the code follows Java rules and is secure.
(c) Runtime Data Area : Memory areas like Heap, stack, Method Area, etc.
(d) Execution Engine : Converts by bytecode into machine code (using interpreter or JIT).
(e) Garbage collector : Frees memory by removing unused objects automatically

- Java is high-level, object-oriented, platform independent programming language. It is widely used for building secure, pro portable and distributed applications.

* Features of Java.
(a) platform Independent : Bytecode runs on any os with JVM.
(b) Object-oriented : Uses objects and classes for structure.

(c) Robust and secure : No pointers: exception handling prevents crashes
(d) Multithreaded : can perform multiple tasks simultaneously
(e) Portable : Can run on various devices without modification.

* Scope of variables
- Scope refer to the part of the program where a variable can be accessed.

(a) Local - Declared inside a method; only accessible there.
(b) Instance : Declared inside a class, but outside methods
(c) static : Declared with static, shared across all objects

* OOP
- is a paradigm based on the concept of 'object' which contain data and methods. It promotes reusability, modularity and scalability.

(a) class
- class is way of representing real world entity in the software domain.
- class is a template of creation of object
- There is no memory associated with the class hence it cannot hold any values.

(b) Object
- Instance of a class
ex. Car c = new Car();

(c) Inheritance
- One class inherits from another 'is-a' kind of hierarchy
ex class Dog extends Animal()
two terms Generalization and Specialization.

(d) Polymorphism
- One method, many forms
- overloading and overriding

(e) Encapsulation
- Hide data using private fields
ex. getters and setters

(f) Abstraction
- Hides implementation, shows only functionality
- abstract class
- Process of identifying key aspects of an entity and ignoring the rest
- Process of extracting essential / relevant details of an entity from the real world.

* Generalization
- Factoring out common elements within a set of categories into a more general category called
- moving up in hierarchy
* specialization: - allows capturing of specific feature of a set of object
- moving down in the hierarchy

* Java primitive data types.
- Java is statically-typed.
- variable must be declared before used

Byte - 1byte
Short - 2byte
int - 4byte
Long - 8byte
Float - 4byte
Double - 8byte
char - 2byte
boolean - 1 bit
- stored directly in the stack memory (when declared inside methods)
- Their values are stored by values, not by reference.

* Non-primitive Data type.
- are objects and ~~are~~ arrays like string, ArrayList, custom classes, etc.
- Reference variable is stored in stack
- Actual object / data is stored in the heap.
- When accessed, stack variable points to the object in heap.

* Java Memory management
- In java, when you use the 'new' keyword, memory is allocated on the heap for the object.

- The Garbage collector (GC) automatically removes objects that are no longer referenced, freeing memory for new objects.
- When a program runs, memory is divided into
  - code segment: contains compiled bytecode or machine instructions
  - Data segment: contains static variables, constants.
- When we create an object, its stored in the heap memory
- Heap is shared among all thread and used for dynamic memory allocation.

- Java uses a generational garbage collection model, where the heap is divided as:

(a) Young generation (Nursery)
  - for new, short-lived objects

(b) Old generation (Tenured)
  - For long-living objects that survive GC.

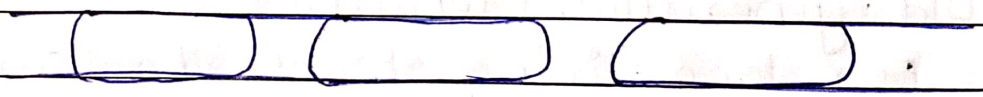
- JVM automatically triggers Garbage Collection when memory is low or full

* Mark and Sweep/compact Method.
- is a Garbage collection algorithm used in Java to automatically manage memory. It works in two phase.

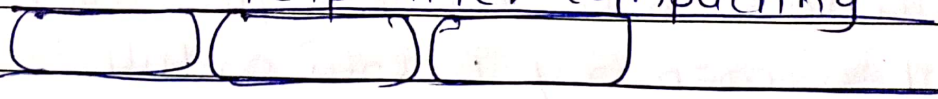
(a) Mark phase:
- The garbage collector starts from GC Roots (like static variables, local variables, etc).
- Identify all objects that are reachable from Java threads, native handles and other root sources as well as the objects that are reachable from these objects and so forth.

(b) Sweep / compact phase.
- Heap is traversed to find the gaps betn the live objects. These gaps are recorded in a free list.
- Move all the live objects to the bottom of the heap, leaving free space at top.

Fragmented heap



Heap after compacting



* Operators in Java

a) Arithmetic Operator
(+) Addition, (-) Subtraction, (*) Multiplication, /(Division), (%)(Modulus, (++) Increment (--) Decrement.

b) Relational Operators

(a) == (equal to)
+ != (not equal to)
> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)

c) Bitwise operator.

IF a = 60 b = 13
then a = 0011 1100
b = 0000 1101

a & b = 0000 1100
a | b = 0011 1101
a ^ b = 0011 0001
~a = 1100 0011

d) Logical operators

&& (logical and), || (logical or), ! (logical not)

e) Assignment Operators

- = → simple assignment
+= → ex. C+=A ⇒ C=C+A
-= → ex C-=A = C=C-A
*=
/=
%=

* Control loop

(a) While loop
- Repeats as long as the condition is true
- condition is checked before execution

syntax.

```
while (condition) {

}
```

(b) do.. while
- Executes the block at least once, then repeats while the condition is true.

syntax

```
do{

} while (condition);
```

(c) For loop
- used when the number of iterations is known
Has initializatn, condn and increment in one line.

```
for (int i=0, i<5; i++)
  {
  }
```

* Conditional Statement

1. If Statement :- Executes a block only if the condition is true.

ex if (condition) {
}

2. if .. else statement
- Executes one block if true, another block if False.

syntax:
```
if (condition)
  {
  }
else
  {
  }
```

3. nested if--else
- if or if--else statement inside another if/else block.

syntax:
```
if (condition1) {
    if (condition2) {
    } else {
    }
} else {
}
```

4. Switch statement
- checks the value of a variable and runs the matching case

syntax.
```
switch (expression) {
  case val1:
      break;
  default:
```

* Access Specifiers

- Data Encapsulation is achieved using access specifiers.

- Specifier — Scope
  - Public → Visible to all
  - Default → All classes within a package
  - Protected → Classes in hierarchy regardless of package
  - Private → Only within the same class

* Java Methods
- collection of statements that are grouped together to perform an operation

syntax.

```
modifier returnType nameOfMethod ( Parameter lis
{
}
```

* ~~Method calling~~
* Accessor method is used to return the value of a private field
- A mutator method is used to set a value of a private field.
- also known as setter and getter method

* Constructor
- A special member function used to initialize the values of the attributes of an object.
- this function is implicitly called when object is created.
- Constructor name must be same as class name.
- There is no return type
- no constructed can be overloaded

* 'this' Reference
- 'this' points to the current object. It holds the reference of the current object

* Static variables

- A static variable is a class-level variable that is shared among all instances (objects) of the class. It is declared using the keyword static.

Note :- Only one copy of a static variable exists in memory, no matter how many objects are created.

* Static method
- A static method belongs to the class rather than to any specific object. It can be called without creating an object of the class.
- Declared using the static keyword.

* The main() method.

- main() method is static method.
- It is called before instantiation of class
- This method has array of string as an argument called as command line agrument
- Any initial information can be passed to a class through this array

* toString method
- toString() returns a string representation of the object.

* varArgs.
- short form of variable arguments
- allows you to pass zero or more arguments of the same types to a method
- Insted of defining multiple overloaded methods, you can use one method with ... (three dots) to accept multiple arguments)

Syntax

returnType methodName (type ... variableName)
{
}

Note- only var one varArgs
- Must be the last parameter

* Arrays
- Arrays references are stored on stack where as actual array is stored on heap.
- int arr[];
  arr = new int[10];
  or
  int[] num = new int[5];
  or
  int[] num = {10, 20, 30, 40, 50};

* Java packages
- Is a group of similar types of classes, interfaces and sub-packages.
- Advantages
  - Used to categorize the classes and interfaces so that they can easily maintained.
  - provides access protection
  - Removes naming collision

* Import and classpath
- To make other classes available to the class you are writing import statement is used
- The .class files are kept in the subdirectories for packages.
- Java virtual Machine will know where to find your compiled class looking at the import statement, however the root directory must be a part of classpath.

* To Wrapper classes
- Java views everything as an object
- Wrapper classes provide a used to convert corresponding data type into an object.
- Wrapper classes include methods to unwrap the object and give back the data type

ex.
```
int x = 20;
Integer i = Integer.valueOf(x);
```

↑ Primitive to Wrapper

```
int y = i.intValue();
```

↑ Wrapper to primitive

* Auto-Boxing and Unboxing

- Auto-Boxing is the process by which Java automatically converts a primitive data type into its corresponding wrapper class object

ex.
```
int num = 10;
Integer obj = num;
```

- Unboxing is the reverse process - Java automatically converts a wrapper class object back to its corresponding primitive type.

ex.
```
Integer obj = 20;
int num = obj;
```

* string class
- can be created:
  - By string literal
  - By new keyword

eg.

```
String s = "Welcome";
String s = new String("Welcome");
```

- String are immutable; the content of a string cannot be changed once it is instantiated
- String constant pool: JVM checks if string already exists in the pool, a reference to the pooled instance is returned; else a new string instance is created and placed in the pool.
- New keyword: JVM will create a new String object in normal heap memory

* StringBuffer class
- Java StringBuffer class is used to created mutable (modifiable) string.
- StringBuffer class is thread safe. It avoids data conflicts but decreases.

```
StringBuffer sb = new StringBuffer()
StringBuffer sb = new StringBuffer(string str)
StringBuffer sb = new StringBuffer(int capacity)
```

In Java, parent class constructor is called first and then child class constructor.

Date / /
Page L&W

* Inheritance
- All variables and method can be inherit from another class except private fields
* Super keyword.
- member initialization of base class in derived class is achieved by using 'super' keyword.

Imp - Constructor of derived class first invokes the constructor of super class.
- If explicit call to parameterized constructor is not given default no argument constructor is called.

Types.

1. single Inheritance
- One subclass inherits from one superclass

2. Multilevel Inheritance.
- A class inherits from a derived class (chain)

3. Hierarchical Inheritance
- Multiple classes inherit from a single parent

4. Multiple Inheritance (via Interface)
- If two classes have the same method and a third class tries to inherit both, Java won't know which method to use this causes ambiguity. Hence use Interfaces.

Note :- Overriden methods should have same argument list of identical type and order else they are considered as overloaded methods.

Date / /
Page L&W

* Polymorphism
- Allows diff objects to share the same external interface although the implementation may be diff
- can be achieved in two ways
(a) compile-time Polymorphism
- Static Binding / Early Binding
- Also called mether overloading
- method to be executed is decided at compile time by the compiler

(b) Run-time Polymorphism (Dynamic Binding)
- Late Binding
- also called method overriding
- method to be executed ~~at~~ is decided at runtime, depending on the object's actual class (even if the reference is of parent type).
- Return type of overriden method must be same else compiler generates an error.

* Upcasting and Downcasting
- upcasting is a casting a child class object to a parent class reference.
- Down done automatically by Java. (Implicit)
- safe and allowed in Inheritance.

ex. class Animal { }
class Dog extends Animal { }

Animal a = new Dog();

Note :- You can only access parent class methods.

b) Downcasting is casting a parent class reference back to a child reference.
- Explicitly required (manual cast)
- Can cause ClassCastException if not done safely

eg. Animal a = new Dog();
Dog d = (Dog) a;

* Covariant Return Type.
- If the overridden method in the subclass returns a subtype (child class) of the return type of the method in the parent class, it is called a covariant return Type.

* 'Final' method and classes.
- final methods cannot be overriden in a subclass.
- All methods in a final class are final by default. A final class cannot be sub classed.
- 'private' methods are final by default
- 'String' class is a final class

* Abstract Class
- A class which contain generic/column features that multiple derived classes can share.
- A abstract method is a method that is declared without an implementation ~~{};~~
- A class has at least abstract method is called abstract class. other methods of the class can be abstract / non abstract.

- Abstract class cannot be instantiated
- A class inheriting from an abstract class must provide implementation to all the abstract methods, else the class should be declared as abstract.
- Abstract modifier cannot be used for constructor

* Interface
- An interface in Java is a fully abstract blueprint of a class. It contains abstract methods and constants. Classes implement interfaces to provide actual behavior.

* Object class
- 'object' class is cosmic super class.
- Every class in Java has Object as the super class, i.e every class in Java implicitly extends Object.

- Object class Methods

• toString() :- Returns a string representation of an object
• boolean equals(Object obj):- Indicates weather some other object is equal to this one
• void finalize(): called by the garbage collector on an object when garbage collector determines that there are no more references to the object.

Note:- If o1 equals(o2) then o1.hashcode() == o2.hashcode() always be true.
- If o1.hashcode() == o2.hashcode is true, it doesn't mean that o1.equals(o2) will always be true

Date / /
Page L&W

- int hashCode() - Returns a hashCode value of an object
- void wait() :- Causes the current thread to wait until another thread invokes the notify() / notifyAll() method for this Object
- notify() - Wakes up a single thread that is waiting on this object's monitor
- notifyAll() :- Wakes up all threads that are waiting on this object's monitor

* Exception-handling
- mechanism that provides a way to respond to run time errors by transferring control to special code called handler.
- Achieved using 'try-catch' blocks.
- Single try - Multiple catch blocks to handle individual errors.
- 'try' block executes normally - unless exception occurs.
- When exception occurs, the system searches for the nearest catch block which matches the type of exception and handles it.

- 'Finally' block.
- A Finally block is always executed where there is an exception or not

* 'throw' :- used to explicitly throw an exception
- used for both checked and unchecked exceptions
- Mainly used to throw custom exception.

Date / /
Page L&W

ex. throw new IOException ("Sorry device error")

* 'throws' keyword
- Exception propagation - Exception is thrown from the top, drops down the callstack until it is caught.
- If a method that is implementing a particular buisness logic doesn't want to handle the exception it throws it to the calling method
- 'throw*' keyword used to declare an exception
- 'throws' is mandatory in case of checked exceptions otherwise compiler will give an error.

Interview

throw	throws
used to actually throw an exception	used to declare that a method ~~and~~ may throw

* Checked Exception - Exceptions that are checked at compile time and must be either handled or declared using throws (e.g. IOException, SQLException).
- Unchecked Exception:- Exceptions that occur at runtime and are not checked at compile time (e.g., NullPointerException, ArithmeticException).

* Collections
- is a framework that provides an architecture to store and manipulate the group of objects.

In list Inference - Duplicates are allowed
- order is fixed.

Date / /
Page L&W

- Operations such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections

* List, List<E> Interface
- Represents a collection of objects that can be accessed by an index.
- classes : ArrayList, LinkedList, Vector
- ArrayList : Extends AbstractList class and implements List interface. Uses a dynamic array for storing the elements.
- LinkedList : Another class that implements List interface, internally uses doubly linkedlist to store the elements

* set, Set<E> interface
- Represents a collection that contains no duplicates
- In Set interface order is not fixed.
- classes : HashSet (unordered), LinkedHashSet (ordered), TreeSet (ascending order)

* Map Interface.

- contains value based on the key i.e key and value pair. Eg
- Map contains only unique elements
- classes : HashMap (unordered), LinkedHashMap (ordered), TreeMap (ascending order).

Date / /
Page L&W

* Sorting of collection objects.

- collections class provide static methods for sorting the elements of collection:
ex. public void sort (List list) is the method used to sort List element.
- If collection elements are of Set type, we can use Treeset.

- Collections.sort() method provides sorting of List elements that are comparable.
- comparable Interface allows sorting of objects of user-defined class.
- Provides public int compareTo (Object obj) method to compare objects of user-defined class.

ex. public int compare (Object obj1, Object obj2)
- compare the first object with second object

- public void sort (List list, comparator c):
- is used to sort the elements of List by the given comparator.

* Generic class
- A Generic class is a class that can operate on objects of any type, specified using type parameters (<T>) at the time of object creation.

- Why Use Generics
  - Type safety
  - code reusability
  - Avoids type casting

syntax:

```
class Box<T>{
    T value;
    void setvalue(T value){
        this.value = value;
    }
    T getvalue(){
        return value;
    }
}
```

- Wildcards in Generic programming represents unknown type - represented by <?>

- Unbounded wildcards <?> Accepts any type.
- Upper Bounded <? extends Type> Accepts Type or any of its subclasses
- Lower Bounded <? super Type> Accepts Type or any of its superclasses

* Lambda Expression
  - Streams and Lambda are two key factors in functional programming
  - Such programs are easier to parallelize providing user the advantage of multi-core architecture to enhance performance.

- Lambda is essentially an anonymous method, where left hand side is parameter list and right hand side looks like method body
- Lambda expressions allows us to create methods that can be treated as data.

* Functional Interface
- A functional interface is an interface that contains exactly one abstract method [may contain default [static methods], also called as Single Abstract Method [SAM] interface.

- Lambda expressions can be used anywhere functional interfaces are expected.
- Lambda can be specified only in the context where a Target type is defined
- This is determined by the lambda's target-type i.e. type of Functional interface type that the lambda implements

NumTest ispositive = (n) -> n >= 0;
Here compiler determines, the target type is interface NumTest. Lambda is assigned to a variable that can later be used to invoke abstract method of NumTest

- Lambda can be used in any context that provides a target type

- One of the context is any context that provides a target type
- One of the context is when lambda is passed as an argument to another method.
- This is most common use of lambda as it gives us a way to pass executable code as an argument to a method.

```
map((x) → x*2)
```

* Method Reference.
- provides a way to refer to a method without executing it.
- The method that accepts another method as reference must have functional interface as parameter.
- The lambda expression is written in place of actual method code. The lambda must be compatible with function inteface

- To create a static method reference
  Classname :: method_name

- To create an instance method reference
  Objref :: method_name

* Imperative programming : You tell the compuder how to do something, step-by-step (focus on control flow).

- Declarative programming: You tell the computer what you want done, not how to do it (focus on result).

* Streams.
- Sequence of elements of elements on which we perform a specific task
- A stream operates on data source such as an array or collection
- Stream classes support declarative programming

* Stream API
- The stream API defines several stream interfaces, packaged in java.util.streams
- Several types of streams are derived from the base stream like - most generic is Stream.
- Stream is a stream of reference types [objects]
- Stream API also provides streams to operate on primitive data - IntStream, LongStream and DoubleStream
- collections → datastructure
  streams → use top operate on the data stored in the collections of array.

* Sequential stream:
Process elements one by one in a single thread, in order of the stream.

- Streams can be obtained for a collection as well as for an array.

Date / /
Page L&W

- Parallel Stream :- Process elements simultaneously using multiple threads, aiming for faster performance on large data.

* collecting.
- The collect() method is a terminal operation in Java stream API used to accumulate elements of the stream into a collection or result (like a List, Set, Map or even a String)

* Performing Stream operation.
- Intermediate operations use lazy evaluation
- Terminal operations use egar evaluation.
- Also some intermediate operations are Stateless while other are stateful.
- In stateless operation each element is process independently of other, in stateful operation such processing might depend on other element values

Intermediate	Terminal
Filter	forEach
Map	reduce
Distinct	count
limit	min
Sorted	max
	Collect

* LocalDate class
- Gives dates and/or time of a day without any time zone information

LocalDate today = LocalDate.now()
localDate adate = LocalDate.of(2023, Month.
AUGUST, 10);

- plusDays(), plusWeeks(), plusMonths(), plusYears() methods can be used to add specific value to the date.

LocalDate nextweek = today.plusWeeks(1)

* Local Time Class
- A localTime represents a time of the day.
- A time Instance can be created using methods now or of

LocalTime current = LocalTime.now();
LocalTime later = LocalTime.of(22,00);

* Zoned Time
- Zone in which a date occurs is an important aspect of ensuring your code is correct.
- IANA - Internet Assigned Number Authority keeps database of all known time zones around the world.
- Each time zone has an ID, which can be found out by ZoneId.getAvailableIds()

* Formatting and Parsing
- The DateTimeformatter class provides three kinds of formatters to print a date/time value.
  - predefined standard formatters
  - Local-specific formatters
  - Formatters with custom patterns

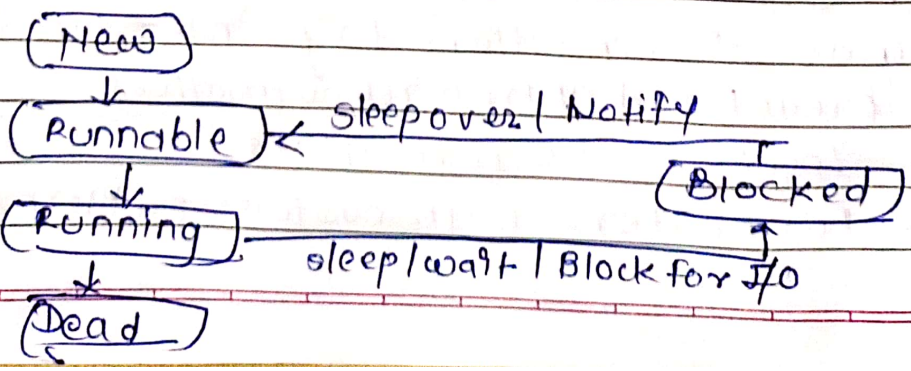
- The format() method can be used to give standard formatter
- custom pattern can be given using a user defined pattern with the help of ofPattern() method.

* Multithreading

- Multithreaded program :- two or more parts of the program can run concurrently, each part is called a thread.
- In multi-threaded environment idle time is minimized because another thread can run when one is waiting.

Multiprocessing	vs	Multithreading
- Allows multiple programs to run concurrently		- multiple parts of a program can be run concurrently
- Each process requires its own separate address space		- Threads share same address space

* Java thread Model



* CPU scheduling.

- Whenever the CPU becomes idle the OS must select one of the processes in the ready queue to be executed. The selection process is carried out by the CPU Scheduler)

- Dispatcher: Module that gives control of the CPU to the process selected by the CPU scheduler.

- Preemptive:- allows the operating system to interrupt a running process and allocate the CPU to another process, often based on priority or time slices.

- Non-preemptive - means that once a process starts executing, it continues to hold the CPU until it either completes its execution or voluntarily enters a waiting state (e.g. for I/O).

* Synchronization
- means making one instruction run before another
- Synchronization is implemented using concept of monitor - An object used as a mutually exclusive lock
- When one thread is inside the synchronized method, all other threads trying to call it on the same instance have to wait for the first one to return

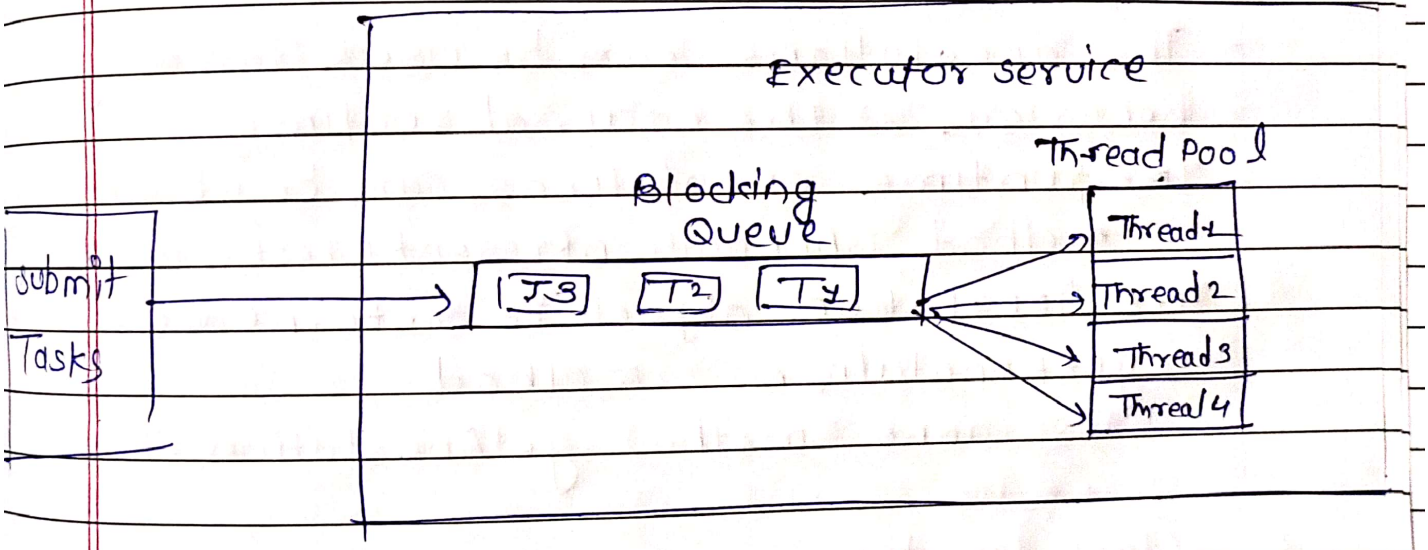
- Another way is to put the calls to any method inside a synchronized block, and pass the object reference of the object being synchronized
- To avoid polling, Java implements inter process communication using wait, notify and notifyAll methods
- These methods can be called only from within a synchronized context.

- wait() :- Tells the calling thread to give up monitor and go to sleep until another thread enters the same monitor.
- notify - wakes up the thread that called wait on the same object
- notifyAll - wakes up all the threads that called wait on the same object. One of the threads will be granted access

* Concurrent Collections

- Concurrent collection represent set of classes that allow multiple threads to access and modify a collection concurrently
- These collections provide thread safe implementations of the traditional collection interfaces like List, Set and Map.

* Executors
- The executors manage thread execution
- The executor framework creates a thread pool with fixed no of thread which will execute the tasks concurrently.
- So instead of creating a new thread every time, we submit multiple tasks which will be executed by the threads in the fixed thread pool.
- The thread pool uses a thread safe data structure - blocked queue to store the tasks.



* Java Reflection
- Reflection is a java API that allows us examine and / or modify behaviour of methods, classes and interfaces at run time.
- Reflection gives us information about the class to which an object belongs and also the methods of that class that can be executed by using the object.

Date / /
Page L&W

* Runtime class of an object
- The getclass() method of object class returns the runtime class of given object.
ex. Employee emp = new Employee()
class C = emp.getClass()

* Annotations
- Annotations are created using @ that precedes a keyword interface
- Annotation consist only method declaration no body
- The annotations can be used to modify behaviour of the code at runtime
- The runtime annotations can be obtained by method. isAnnotationPresent(Test.class)
- The Annotation object is returned by using getAnnotation() method
Test anno = method.getAnnotation(Test.class)

* Class Loader
- three phases :- load, link, Initialize
- load phase is responsible for loading the class file from physical memory/network
- link phase :- is further divided as verify, prepare and resolve
  - verify :- checks if given byte code is valid?
  - prepare :- static variables are assigned the memory [with default value]

- resolve :- all symbolic references in the class are resolved
- References: to other classes, constant pool are changed from symbolic names to actual reference.
- In Initialize phase :- static initialization block get executed

* Node.js

- Node.js is an open-source, cross-platform JavaScript runtime environment that lets developers execute JavaScript code outside of a web browser. It's built on chrome's V8 JavaScript engine.

- Browser Javascript :
  - Runs directly within a web browser (like chrome, Firefox, safari)
  - Primarily used for front-end web development enabling interactive and dynamic user interfaces.
  - Has access to the Document Object Model (DOM) to manipulate web page content
  - cannot directly access the user's file system or OS resources.
  - Execution is often event-driven, responding to user actions

- Node.js:
  - Runs on a server-side environment or as a standalone application
  - Primarily used for back-end web development, building APIs, microservices, and server-side applications.
  - No access to the DOM as there's no browser involved.
  - Can directly access the file system, network, and other OS resources.
  - Ideal for I/O-intensive operations and real-time applications due to its non-blocking, event-driven architecture.
  - Uses (NPM) Node Package manages for managing and sharing packages.

* Node.js REPL
  - The node.js REPL (Read-Eval-Print Loop) is an interactive shell environment that allows you to execute JavaScript code snippets directly and see the results immediately.

It's an excellent tool for:

- Experimenting with Node.js features and Javascript syntax.
- Testing small code snippets without creating a file
- Debugging and understanding how certain functions or modules behave.

* Introduction to spring framework.

- The Spring Framework is like a swiss Army knife for Java developers. It's a powerful, open-source framework that helps you build all sorts of Java applications faster and more efficiently. Think of it as a set of tools and guidelines that make creating robust, scalable, and easy-to maintain applications much simpler. Its main goal is to make Java development easier by handling a lot of the complex setup and common tasks for you.

* Architecture.

- Spring's architecture is modular, means it made up of many independent parts you can pick and choose from. At its core, Spring uses something called an Inversion of Control (IoC) containes.

- IOC (Inversion of control): This is term in which spring takes control of creating and managing your application's object. Insted of you telling an object tocreate its dependencies (other objects it needed), Spring injects those dependencies into it.
- Dependency Injection (DI): This is the practical way IoC is achieved. Spring 'injects'

the things an object needs to function, rather than the object finding them itself

- AOP (Aspect-Oriented Programming):
This helps you handle "cross-cutting concerns" like logging or security. Instead of sprinkling logging code everywhere, you can define it once as an 'aspect' and Spring applies it where needed.

* Spring MVC Architecture

- Model :- This holds your data and buisness logic.
- View : This is what the user sees (like an HTML page).
- Controller : This handles user requests, talks to the model, and decides which view to show.