Day 1

Language

```
* C, C++, Java, C#, Python, GO, Ruby etc.
* Data types
* Syntax and Semantics
* Tokens
    1. Identifier
    2. Keyword
    3. Constant / Literals
    4. Operators
    5. Punctuators / Separators
* Built in features
* To implement B.L we should lanaguage.
* Types of application
    1. Console User Interface( CUI )
    2. Graphical User Interface( GUI )
    3. Library Application( .jar )
```

Technology

```
* ASP.NET, Java etc.
* It is used to develop application.
* Every language can be considered as Technology but every Technology can
not be considered as language.
```

Platform

```
* It provides environment in which we can run application.
* Types of platform
    1. Hardware based platform
        - All Operating Systems.
    2. Software Only Platform
        - Java, MS.NET Framework
```

Framework

```
* It is a library of reusable classes which is used to develop
application.
* RMI( Remote Method Invocation)
```

```
     - It is distributed application development Framework.
* AWT, Swing
     - It is GUI application development Framework.
* Collection Framework
     - It is library of Data structure and algorithms.
* Struts
     - It is MVC based web application development Framework
* Hibernate
     - It is automatic persistance Framework.
* JUNIT
     - It is a testing framework
```

Java language is both Technology as well as Platform.

Java History

```
* Core Java Vol - 1 ( Cay Horstman )
* It is developed in 1991 at Sun Micro Systems.
* It is invented By James Gosling and his Team ( Green Team).
* Initial Name of Java was "Oak". But due to name ambiguity it is renamed
to "Java". It is name of coffee.
* First version of java(1.0) was relased in 1996.
* Now it is product of Oracle.
* Current version of java is : Java 13
```

Java Platforms

```
1. Java SE
2. Java EE
3. Java ME
4. Java FX
```

Java SE

```
* Java Standard Edition.
* It is also called as core java.
* It is used to develop CUI, GUI, networking application, distributed
application, libraries etc.
* Java SE API's are sub set of Java EE API.
```

Java EE / JEE

```
* Java Enterprise Edition
* It is also called as Web Java/Enterprise Java / Advanced Java.
* It is used to develop web application and web services.
* Java EE API's are super set of Java SE and ME API's
```

Java ME

```
* Java Micro Edition
* It is used to develop application for consumer devices.
* e.g Mobile
* It is sub set of Java SE API.
```

Java Fx

```
* It is used to develop rich GUI application for internet then we should
use Java Fx.
```

Java SE Platform / Core Java

```
* Java Platform consist of two components
    1. Java Application Programming Interface( Java API ).
    2. Java Virtual Machine( JVM ).
* In java, interface/class/enum is generally refered as Java API.
* "rt.jar" is a java library file which contains implementation of Java
API.
* JVM is runtime environment of java which is used to execute any java
applicattion.
* It is also called as abstract computer.
* rt.jar and JVM are integral part of JRE( Java Runtime environment ).
* JRE is a software.
* If we want to deploy java application on client's machine then we must
install JRE on that machine.
```

SDK = Lang Tools + Documentation + Supporting Libs + Runtime environment.

JDK = Java Lang Tools(bin) + Java Docs (docs) + rt.jar + JVM

JDK = Java Lang Tools + Java Docs + JRE[rt.jar + JVM]

jdk 1.8 is a software which supports all the features specified in Java SE 8.

JDK Installation Directory Structure

1. bin
    - It contains java language development tools.
2. include
    - In context of java, C/C++ code is called native code.
    - JNI : Java Native Interface, it is java's framework used to access native code in java.
    - Using JNI, if we want to access native code in java then we need to use header files declared in include directory.
    - JNI : reference : "www.artima.com"
3. lib
    - It contains jar files required for third party tools
    - e.g IDE, Build tools
4. src (src.zip)
    - It contains source code of Java API.
5. jre
    - It contains JVM implementation and rt.jar
6. docs
    - It contains documentation of Java API
7. man
    - It contains documentation of java language tools.

Type

```
It is general term used in java to refer built in type as well as user
defined type.
```

Organization of types:

```
* All the types are organized in jar file.
* Jar File Contains
    1. Package(s)
    2. Menifest File ( Binary File )
* Package can contain
    1. Sub Package
    2. Interface
    3. Class
    4. Enum
    5. Error
    6. Exception
    7. Annotation
* Interface can contain
    1. Final Field / constant
    2. Abstract method
    3. Default method
    4. Static method
    5. Nested Interface
* Class Can contain
```

```
    1. Nested Type( interface /class/enum )
    2. Field
    3. Constructor
    4. Method
```

Java Terminology

```
*   C++ :   Java
* class  :   class
* Object   :    Instance
* Base class    :   Super class
* Derived class :   Sub class
* Access Specifier  :   Access Modifier
* Namespace     :   Package
* Data Member   :   Field
* Member Function   :   Method
* this pointer  :   this reference
* Concrete class : We can instantiate concrete class
* Abstract class : We can not instantiate abstract class
* Final class : We can not extend final class.
```

Naming/Coding Conventions

1. Camel Case Naming Convention
2. Pascal Case Naming Convention

Camel Case

```
* e.g
    1. main()
    2. parseInt()
    3. showInputDialog( )
* In this case, except first word, first character of each word must be in
upper case.
* In java, it is used for
    1. Field name
    2. Method name
    3. Method parameter and local variable
    4. Object reference.
```

Pascal Case

```
* e.g.
    1. System
    2. StringBuilder
    3. NullPointerException
    4. IndexOutOfBoundsException
* In this case, including first word, first character of each word must be
in upper case.
* In java, it is used for
    1. Type Name( Interface/class/Enum )
    2. File Name
```

Name of the package should be in lower case.

Name of final field and enum constant must be in upper case.

- src.zip : It constains source code of java API.
- rt.jar : It contains compiled code of java API
- java docs : It contains documentation of Java API.
- rt.jar file contains following "main packages":
  1. com
  2. java
  3. javax
  4. org
  5. sun
- java main package contains 14 "sub" packages
  1. applet
  2. awt
  3. beans
  4. io
  5. lang
  6. math
  7. net
  8. nio
  9. rmi
  10. security
  11. sql
  12. text
  13. time
  14. util
- If we want to group functionally equivalent / related types together then we should use package.
- java.lang package contains all the fundamental types of core java.

"Hello World!!" Application

```
 Step 1 : vim Hello.java
 Step 2 :
```

```
class Program
{
 public static void main( String[] args )
    {
        System.out.println("Hello World!!");
    }
}
```

```
Step 3: javac Hello.java ( Press enter )
Step 4: java Program
* Java Compiler generates .class file. It contains "bytecode".
* Bytecode is object oriented assembly language code designed for JVM.
* Using "javap -c" option we can see bytecode.
* eg.   javap -c Program.class
* Executing bytecode is a job of JVM.
```

System

```
* It is a final class declared in java.lang package.
* Following are fields of System class
    1. System.in
    2. System.out
    3. System.err
```

Definition of System class

```
package java.lang;
public final class System
{
    //Fields
    public final static InputStream in;
    public final static PrintStream out;
    public final static PrintStream err;
    //Methods
    public static Console console(){   }
    public static void gc(){   }
    public static void exit(int status){   }
}
```

System.out

```
* out is object reference / reference of java.io.PrintStream class.
* out is declared as public static final field inside System class.
```

println() method

```
* print(), printf() and println() are non static methods of
java.io.PrintStream class.
```

Day 2

Data Type

```
* Data describes 3 things
    1. Memory : how much memory is required to store the data.
    2. Nature : which type of data can be stored in memory
    3. Operation : Which Operations can be performed on the data.
* Types of data type
    1. Primitive Data type
    2. Non Primitive Data Type
```

Primitive Data Types(8)

```
* It is also called as value type.
* There are 8 Primitive/value types in java.
* variable / instance of primitive/value type get space on stack segment.
```

```
class Program
{
    public static void main(String[] args)
    {
        int number = 10;
    }
}
```

```
* Following are primitive types
    1. boolean  Not Specified.
    2. byte     ( 1 byte )
    3. char     ( 2 bytes )
    4. short    ( 2 bytes )
    5. int      ( 4 bytes )
    6. float    ( 4 bytes )
    7. long     ( 8 bytes )
    8. double   ( 8 bytes)
* In Java, primitive types are not classes. e.g int is not class.
```

Wrapper class

```
* For every pritive type java has given a class. It is called "Wrapper
class".
* e.g
    int : primitive type
    Integer : Wrapper class
1. boolean  :   java.lang.Boolean
2. byte     :   java.lang.Byte
3. char     :   java.lang.Character
4. short    :   java.lang.Short
5. int      :   java.lang.Integer
6. float    :   java.lang.Float
7. long     :   java.lang.Long
8. double   :   java.lang.Double
* All the wrapper classes are final.
```

Default values

```
1. boolean  :   false
2. byte     :   0
3. char     :   +u0000
4. short    :   0
5. int      :   0
6. float    :   0.0f
7. long     :   0L
8. double   :   0.0d
* In general, variable of value type, by default contains 0 value.
```

Non Primitive Data Types(4)

```
* It is also called as reference type.
* There are 4 non primitive/reference type in java
    1. Interface
    2. Class
    3. Enum
    4. Array
* Instance of non primitive / reference type get space on Heap section.
```

- In java, if we want to use any local variable then it is mandatory to store value inside it.

```
class Program
{
    public static void main(String[] args)
    {
        int x;  //OK
        System.out.println(x);  //Error

        int y = 10;  //OK
        System.out.println(y);  //Ok

        int z;  //Ok
        z = 20; //OK
        System.out.println(z);  //Ok
    }
}
```

Widening

```
* Process of converting state/value of variable of narrower type into
wider type is called widening.
```

```
int num1 = 10;
//double num2 = ( double )num1;   //OK
double num2 = num1; //OK : Widening
```

```
* In case of widening, explicit type casting is optional.
```

Narrowing

```
* Process of converting state/value of variable of wider type into
narrower type is called Narrowing.
```

```
double num1 = 10.5d;
int num2 = ( int )num1; //OK : Narrowing
//int num2 = num1;    //Not Ok
```

```
* In case of Narrowing, explicit type casting is mandatory.
```

Boxing

```
* Process of converting state/value of variable of primitive/value type
into non primitive/reference type is called boxing.
```

```
int number = 10;
//Boxing
String strNumber = Integer.toString(number);
```

```
* valueOf() is static method of String class which is used to convert
state of variable of value type into String.
```

```
int number = 10;
//Boxing
String strNumber = String.valueOf(number);
```

```
int number = 10;
//Boxing
Integer n1 = Integer.valueOf(number);
```

UnBoxing

```
* Process of converting state/value of instance of non primitive/reference
type into  primitive/value type is called unboxing.
```

```
 String str = "125";
int num1 = Integer.parseInt(str); //UnBoxing
```

```
* parseXXX() is a method of wrapper class which is used to convert string
into primitive type.
* If string does not contain a parsable numeric value then parseXXX()
method throws NumberFormatException.
```

```
String str = "abc";
int num1 = Integer.parseInt(str); //NumberFormatException
```

Java Buzzwords

1. Simple
2. Object Oriented
3. Architecture Neutral
4. Portable
5. Robust
6. Multithreaded
7. Secure
8. Dynamic
9. High Performance
10. Distributed.

Java is Simple

```
* Java language is derived from C & C++ i.e. Java language follows syntax
of C and Concepts of C++.
* Syntax of java is simpler than C/C++ hence java is simple
    1. No need to include header files
    2. Java do not support structure and union
    3. Java do not support delete operator and destructor
    4. It doesn't support friend function and class
    5. It doesn't support operator overloading
    6. It doesn't support multiple class inheritance hence therse is no
diamond problem and virtual base class.
    7. It doesn't support constructors member initializer list and default
argument
    8. There is no concept separate declaration and definition. Everything
is definition.
    9. We can not define anything globally.
    10. Java do not support casting operators
    11. Java do not support pointer arithmetic.
    12. Java do not support private and protected mode of inheritance.
* Since size of software which is required to develop java application is
small, java is simple.
```

Java is Object Oriented

```
* 4 Major pillars of OOPs
    1. Abstraction
    2. Encapsulation
    3. Modularity
    4. Hierarchy
* 3 minor pillars
    1. Typing
    2. Concurrency
    3. Persistance
* Since java support all major and minor pillars of oops, it is considered
as object oriented.
```

java is Architecture Neutral Language

```
* CPU Architecture's
    1. x86/x64
    2. ARM
    3. Power PC
    4. ALPHA
    5. Sparc
* After compilation, java compiler generates bytecode which is cpu
architecture neutral code which makes java architecture neutral.
```

Java is Portable programming Language

```
* Java is protable because it is architecture neutral.
* Java's Slogan is :
"Write Once Run AnyWhere"
* JVM is platform dependant
* JVM is integral part of JRE hence JRE also platform dependant
* JDK is platform dependant
* bytecode/.class file is architecture neutral
* Size of data types on all the platforms is constant hence java is truely
portable.
    1. boolean  Not Specified.
    2. byte     ( 1 byte )
    3. char     ( 2 bytes )
    4. short    ( 2 bytes )
    5. int      ( 4 bytes )
    6. float    ( 4 bytes )
    7. long     ( 8 bytes )
    8. double   ( 8 bytes)
* Since java is portable, it doesn't support sizeof operator.
```

Java is Robust Programming Language

```
1. Architecture Neutral
    * Since java is architecture neutral, developer need not not to do
hardware / OS specific code.
2. Object Oriented
    * Hierarchy allows developer to reuse existing code which reduces
developers effort, development time and cost.
3. Memory Management
    * In java developer can allocate memory for object but deallocating
that memory is a job of Garbage Collector / Finalizer.
4. Exception handling
```

```
    * Java compiler helps developer to handle exception which reduces
developers effort.
```

Java is multithreaded

```
* Program in execution is called process/task.
* In other words, running instance of a program is called process.
* Light weight process / sub process is called thread.
* In other words, thread is seperate path of execution, which runs
independantly.
* An ability of OS to execute single process at time is called
singletasking.
* e.g. MSDOS
* An ability of OS to execute multiple process at time is called multi
tasking.
* e.g All modern OS.
* It is possible to achive multitasking using process as well as thread.
* If any application uses single thread for execution then it is called
single threaded application.
* If any application uses multiple thread for execution then it is called
multi threaded application.
* When JVM starts execution of java application, it also starts execution
of "main thread" and "Garbage collector". Hence every java application is
multithreaded.
```

Main Thread

```
1. It is Non Deamon / User Thread
2. It is responsible for invoking main() method.
3. It's priority is 5.
```

Garbage Collector

```
1. It is also called as finalizer.
2. It is Daemon / Background Thread.
3. It is responsible for releasing / reclaiming / deallocating memory of
unused objects.
4. It's priority is 8.

* Thread is non java resource ( unmanaged resource ). With the help of
SUN/ORACLE's thread framework, we can use OS thread in java application.
In other words, java has given built in support to thread hence java is
considered as multithreaded.
```

Java is secure programming language.

```
* If we want to develop secure application then we use types declared in
java.security package. It means that SUN/ORACLE has given support to
achive security hence it is considered as secure.
```

Java is dynamic programming language

```
* In java, all the methods are by default virtual.
* We can run java application on any evolving hardware as well as
operating system hence it is truly dynamic
* We can add new types inside existing libraries(.jar) at runtime time.
```

Java is High Performance programming language.

```
* Performance of java is slower than C/C++.
* During conversion of bytecode into native cpu code, JIT compiler use
optimization technique which improves Performance of application.
* With the help of JNI, we can invoke code written in C/C++ into java
which help us to improve Performance of application
```

Java is distributed.

```
* RMI : Remote method invocation.
* It is java language feature which supports Service Oriented
Architecture( SOA).
* Java is distributed because it supports RMI.
```

- If we want to access any type outside package then we should use either F.Q.TypeName or import statement.
- java.lang package is by default imported in every .java file.

Stream

```
* It is an abstraction( object ) which is used to produce( write ) and
consume ( read ) information from source to destination.
* Console = Keyboard + Monitor
* Standard Stream objects of java which is associated with Console:
    1. System.in    :   Keyborad
    2. System.out   :   Monitor
    3. System.err   :   Error Stream
```

Console IO

```
* Scanner is final class declared in java.util package
* If we want to read tokens from Console, File etc, then we can use
Scanner class.
* Syntax:
    import java.util.Scanner;
    Scanner sc = new Scanner(System.in);
* Methods of Scanner class:
    1. public String nextLine()
    2. public int nextInt()
    3. public float nextFloat()
    4. public double nextDouble()
```

Class

```
* It is collection of fields and methods
    class ClassName
    {
        //Fields
        //Methods
    }
* Structure and behavior of object/instance depends of fields and methods
declared inside class hence class is considered as a
template/model/blueprint for instance.
* Class represent collection of instances which is having commong
structure and common behavior.
* e.g Car, Laptop
* If we want to achive Encapsulation then we should define class.
```

Instance

```
* In java, object is called as instance.
* An entity which is having physical existance is called instance.
* If entity has state, behavior and identity then it is called instance.
* e.g Maruti 800, MacBook Air etc.
* If we want to achive abstraction then we should create instance and
perform operations on it.
```

Access modifiers(4)

```
1. private( - )
2. package level private( ~ )
3. protected( # )
4. public( + )
```

Day 3

Java Entry Point

```
* According JVM specification, "main" should be entry point method in java
application.
* Java compiler do not check whether class contains "main" method or not. It is
responsibility of JVM.
* Syntax:
"public static void main(String[] args)"
* With the help of main thread, JVM invoke main method.
* In java, we can overload main method.
* We can define "main" method per class but only one main method can be considered
as entry point method.
```

Object class

```
* Object is non final and concrete class declared in java.lang package.
* In java, every class( not interface ) is directly or indirectly extended from
java.lang.Object class.
* java.lang.Object class do not have super class hence it is called as "Ultimate
base class" / "Super Cosmic Base class" / "Root Of Java Class Hierarchy".
* It is introduced in JDK 1.0
```

Members of java.lang.Object class

```
* It doesn't contain nested type.
* It doesn't contain field.
* It contains only parameterless constructor.
Object o1 = new Object( );  //OK
Object o2 = new Object( 24 ); //Not OK
* It contains 11 method( 5 non final + 6 final methods ).
* In java, we can not override final method.
```

Methods of java.lang.Object class

1. public String toString( );
2. public boolean equals(Object obj);
3. public native int hashCode( );
4. protected native Object clone( ) throws CloneNotSupportedException;

5. protected void finalize( )throws Throwable
6. public final native Class<?> getClass( );
7. public final void wait( ) throws InterruptedException
8. public final native void wait(long timeOut) throws InterruptedException
9. public final void wait(long timeOut, int nanos) throws InterruptedException
10. public final native notify( );
11. public final native notifyAll( );

Java class and Instance

```
* To define class, we should use class keyword.
    class ClassName
    {
        //Fields
        //Methods
    }
* In java class members are by default considered as package level private / default.
* In java, data member is called as field. It can be static or non static.
* Non static field is called as instance variable and static field is called as class-level variable.
```

```java
class Test
{
    private int x;   //Instance variable
    private static int y; //Class-level var.
}
```

```
* If we create instance of a class then only non static field / instance variable get space inside it.
* In java, if we want to create instance of a reference type then it is mandatory to use new operator.
* If we create instance using new operator then it gets space on heap section.
* Instantiation in C++
```

```cpp
    Complex *ptr = new Complex( );
    Complex *ptr = new Complex( 10, 20 );
```

```
* Everything on heap section is anonymous. If we want to perform operations on
instance then first it is mandatory to create reference of it.
* Instantiation in Java
```

```
    Complex c1 = new Complex( );    //Ok

    Complex c2;     //OK
    c2 = new Complex( );    //OK
```

```
* if we want to perform operations on instance then we should define method inside
class.
* Process of calling method on instance is called message passing.
* "this" is implict reference variable which is available in every non static
method of the class that is used to store reference of current instance.
```

Constructor

```
* It is a method of a class which is used to intialize instance.
* Imporant point about constructor
    1. It's name is same as class name
    2. It doesn't have return type
    3. It is designed to call implicitly
    4. It is designed to call once per instance.
* Types
    1. Parameterless ctor
    2. Parameterized ctor
    3. Default ctor.
```

```java
class Complex
{
    private int real, imag;
    //Parameterless constructor
    public Complex( )
    {
        this.real = 10;
        this.imag = 20;
    }
}
Complex c1 = new Complex( );
```

```
class Complex
{
    private int real, imag;
    //Parameterized constructor
    public Complex( int real, int imag )
    {
        this.real = real;
        this.imag = imag;
    }
}
Complex c1 = new Complex( 10, 20 );
```

```
* We can write multiple constructor inside class. It is called constructor
overloading.
* If we want to reuse implementation of existing constructor then we should call
constructor from another constructor.
* If we want to call constructor from another constructor then we should use "this
statement";
* this statement must be first statement in constructor body.
* Process of calling constructor from another constructor is called constructor
chaining.
* To reduce developer's effort we should use constructor chaining.
```

```
class Complex
{
    private int real;
    private int imag;
    public Complex( )
    {
        this( 10, 20 ); //ctor Chaining
    }
    public Complex( int real, int imag )
    {
        this.real = real;
        this.imag = imag;
    }
}
```

```
* we can use any access modifier on constructor.
```

null

```
* It is literal which is used to initialize reference variable.
```

```
    int number = null;  //Not OK
    Complex c1 = null;  //Ok
```

```
* If reference contains null value then it is called null reference variable or
null object.
* In above code c1 is null object.
```

```
    int *ptr = NULL;    //OK
    int number = NULL;  //OK
```

```
* In C++, NULL is a macro whose implicit value is 0.
* It is designed to initialize pointer.
```

NullPointerException

```
* Using null object, if we try to access any member of the class then JVM throws
NullPointerException.
```

```
Complex c1 = null; //OK
c1.printRecord( );  //NullPointerException
```

```
* we can solve it using
```

```
Complex c1 = null; //OK
c1 = new Complex( );    //OK
c1.printRecord( );  //OK
```

```
Complex c1 = new Complex(); //Ok
c1.printRecord( );  //OK
```

Value Type Versus Reference Type

Value Type

```
1. Primitive type is called as value type.
2. Instance of value type get space on stack segment.
3. There are 8 value types in java.
4. Variable of value type contains value.
5. If we assign variable of value type to the another variable of value type then
value gets copied.
    int num1 = 10;
    int num2 = num1;
6. Default value of value type is 0.
7. Variable of value type do not contain null.
    int number = null;  //Not OK
```

Reference Type

```
1. Non Primitive type is called reference type.
2. Instance of reference type get space on heap section.
3. There are 4 reference types in java.
4. Variable of reference type contains reference.
5. If we assign variable of refence type to the another variable of reference type
then reference gets copied.
    Complex c1 = new Complex( 10,20);
    Complex c2 = c1;
6. Default value of reference type is null;
7. Variable of reference type can contain null.
    Complex c1 = null;  //OK
* Local reference variable get space on stack segment.
* If reference is field of the class then it gets space on Heap section.
```

toString() method

```
* It is non final method of java.lang.Object class.
* Syntax:
    public String toString( );
* If we want to retrun state of the object in String format then we should use
toString() method.
* If we do not define toString() method inside class then super class's toString
method gets called.
* toString() method of object class returns String in following format
    "F.Q.ClassName@HashCode"
* According to client's requirement, if implementation of super class method is
partialy complete then we should redefine/override method in sub class.
* toString method should return string which should contain concise and
```

```
informative result.
* Every class should override toString() method.
```

```java
public String toString( )
{
    return String.format("%-15s%-5d%-10.2f",this.name, this.empid, this.salary);
}
```

Path

```
* path is OS platforms environment variable that is used to locate java tools.
* If we want to set path then we should use following command:
    export PATH=/usr/bin/
* Above step is optional in linux.
* To check value of PATH we should use:
    echo $PATH
```

Classpath

```
* It is environment variable of Java Platform, which is used to locate .class file
/ .jar file.
* If we want to set classpath then we should use following command:
    export CLASSPATH=./bin/
* By default, CLASSPATH is set to current directory.
```

Package

```
* It is java language feature, which is used
    1. To avoid name ambiguity
    2. To group functionaly equivalent types together.
* We can not instantiate package.
* package is keyword in java.
* package can contain
    1. Sub Package
    2. Interface
    3. Class
    4. Enum
    5. Error
    6. Exception
    7. Annotation
```

```
namespace std
{
    class Complex
    {

    };
}
```

```
package p1; //OK
class Complex
{

};
```

```
* If we want to define any type inside package then we must write package
declaration statement inside .java file.
* Package declaration statement must be first statement in .java file.
```

```
package p1; //OK
package p2; //Not OK
class Complex
{

};
```

```
* In other words, we can not define class in multiple packages.
* package name is physically mapped to folder.
```

Day 4

Package

- It is a java language feature that is used to
    1. To avoid name collision/clashing/ambiguity.
    2. To group functionally equivalent/related types together.
- It can contain following types:
    1. Sub package
    2. Interface
    3. Class
    4. Enum
    5. Error
    6. Exception
    7. Annotation.
- "rt.jar" contains following main packages:
    1. sun
    2. org
    3. com
    4. java
    5. javax
- java main package contains following sub packagaes:
    1. applet
    2. awt
    3. beans
    4. io
    5. lang
    6. math
    7. net
    8. nio
    9. rmi
    10. security
    11. sql
    12. text
    13. time
    14. util
- How to declare namespace in C++

```
namespace std
{
    class Stack
    {

    };
}
```

- How to declare package

```
package p1;
class Stack
{
    //TODO : Declare member
}
```

- Package declaration statement must be first statement in .java file.

```
package p1; //Ok
package p2; //Not Ok
class Stack
{
    //TODO : Declare member
}
```

- We can not declare any type inside multiple packages.
- If we declare any type inside package then it is called packaged type otherwise it is called as unpackaged type.
- package name is physically mapped to folder/directory.
- If we want to access type from different package then we should use F.Q. Type name or import statement.

```
class Program
{
    public static void main(String[] args)
    {
       //Stack s1 =  new Stack( );//Not OK
       p1.Stack s1 = new p1.Stack(); //Ok
    }
}
```

- Using import

```
//import p1.*;    //OK
import p1.Stack;    //OK
class Program
{
    public static void main(String[] args)
    {
       p1.Stack s1 = new p1.Stack(); //Ok
       Stack s2 = new Stack(); //Ok
    }
}
```

- In java, default access modifier of any type is package level private

```
package p1;
??? class Complex
{   }
// ??? -> package level private
```

- If we want to access type outside package then it must be public.
- Access modifier of a type can be either package level private or public Only. In other words, we can declare type private or protected.

```
package p1;
public class Complex
{   }
```

- According java language specification, name of public class and .java file must be same.
- Commands to compile and execute program

```
javac -d ./bin/ ./src/Complex.java
export CLASSPATH=./bin/
javac -d ./bin/ ./src/Program.java
java Program
```

- java.lang package is by default imported in every .java file hence importing lang package is optional.
- If we define any type without package then it is considered as member of default package.
- Since we can not import default package, it is impossible to access unpackaged type from packaged type. But we can access packaged type from unpackaged type.
- we can define types in different package

```
package p1;
public class Complex
{   }
```

```
package p2;
import p1.Complex;
public class Program
{
    public static void main(String[] args)
    {
        Complex c1 = new Complex();
    }
}
```

- We can define types in same package. In this case use of import statement is optional.

```
package p1;
public class Complex
{   }
```

```
package p1;
//import p1.Complex;  //OK
public class Program
{
    public static void main(String[] args)
    {
        Complex c1 = new Complex();
    }
}
```

- Math is a final class declared in java.lang package.
- It contains fields and methods for performing numeric operations.
- All the members of Math class are static.
- without typename, if we want to access static members then we should use static import statement

```
import static java.lang.System.out;
//import static java.lang.Math.*;
import static java.lang.Math.PI;
import static java.lang.Math.pow;
class Program
{
    public static void main( String[] args )
    {
        float radius = 10.0f;
        //float area = ( float )( Math.PI * Math.pow(radius, 2) );
        float area = ( float )( PI * pow(radius, 2) );
        out.println("Area    :   "+area);
    }
}
```

Static in Java

- According oops, static variable is called class level variable and it must exist at class scope. Hence we can not declare local variable static.
- In java, we can not declare local variable static but we can declare field static.

```
class Test
{
        private static int count; //OK
        public void print( )
        {
        //static int count; //Not Ok
                ++ count;
                System.out.println("Count       :       "+count);
        }
}
```

- If we want to share, value of any field in all the instances of same class then we should declare that field static.
- In java, static field get space during class loading, once per class on method area.
- Non static field is also called as instance variable. It gets space once per instance.
- static field is also called as class level variable. It gets space once per class.
- To access instance members we should use instance whereas to accss class level members we should use class name and dot operator.

```
class Test
{
        int x = 10;     //Instance variable
        static int y = 20;      //Class level variable
}
public class Program
{
        public static void main(String[] args)
        {
                Test t = new Test();
                //System.out.println("X :       "+Test.x );     //Not OK
                System.out.println("X   :       "+t.x );        //OK

                System.out.println("Y   :       "+Test.y);      //Ok
                System.out.println("Y   :       "+t.y); //Ok
        }
}
```

- Non static method is also called as instance method and static method is also called as class level method.
- Instance method is designed to call on instance whereas class level method is designed to call on class name.
- since non static field get space inside instace, we should initialize it inside constructor. JVM invoke constructor, once per instance.
- To intialize static field, we should use static initializer block. JVM invoke it once per class.
- we can write multiple static initializer block inside class.

```
class Test
{
        private int num1;
        private int num2;
        private static int num3;
        static  //static initializer block
        {
                Test.num3 = 0;
        }
        public Test( )
        {
                this( 0, 0);
        }
        public Test( int num1, int num2 )
        {
                this.num1 = num1;
                this.num2 = num2;
    }
}
```

- If we want to access non static members of the class then we should define non static method / instance method inside class.
- instance method is designed to call on instance.
- If we want to access static members of the class then we should define static method/class level method inside class.
- class level method is designed to call on class name.
- Static method do not get this reference.
- this reference is considered as link between non static field and non static method.
- Since static method do not get this reference, we can not access non static members inside static method directly.
- In other words, static method can access static members of the class only.
- Using instance, we can access non static members inside static method

```
public class Program
{
        private int num1 = 10;
        private static int num2 = 20;
        public static void main(String[] args)
        {
                //System.out.println("Num1      :       "+num1);//Not OK
                Program p = new Program();
                System.out.println("Num1        :       "+p.num1);//Ok
                System.out.println("Num2        :       "+num2); //OK
    }
}
```

- We can not declate local variable and top level class static.

Singleton class

- It is a creational design pattern
- It allows us to create only one instance.

```
class Singleton
{
        private Singleton( )
        {       }
        static Singleton instance;
        public static Singleton getInstance( )
        {
                if( instance == null )
                        instance = new Singleton();
                return instance;
        }
}
```

Final

- After storing value, if we dont want to modify state/value of the variable then we should use final keyword.

```
final int number = 10;
++ number;  //Not OK
System.out.println( number );
```

```
final int number = 10;  //OK
System.out.println( number ); //OK
```

```
final int number;  //OK
number = 10;    //OK
System.out.println( number ); //OK
```

```
final int number = 10;  //OK
number = 20;//Not OK
System.out.println( number );
```

- We can provide value to the final variable at runtime.

```
final int number;
System.out.print("Number       :       ");
```

```
number = sc.nextInt();
```

- If we dont want to modify state field inside any method of the class(including ctor) then we should declare field final.
- Note : If we want to declare field final then we should declare it static also.

```
class Math
{
        public static final double PI = 3.14;
}
```

- In java, we can not delare instance final but we can declare reference final.

```
final Complex c1 = new Complex(10, 20); //OK
c1.setReal(11); //Ok
c1.setImag(22); //Ok
//c1 = new Complex(50, 60); //Not OK
System.out.println(c1.toString());
```

Array

- It is linear data structure which is used to store elements of same type in continous memory location.
- If we want to access elements of array then it is necessary to use index and subscript operator.
- Array index always begins with 0.
- In java, array is reference type i.e array instance get space on heap section.
- Types of array
    1. Single dimensional
    2. Multi dimensional
    3. Ragged array
- Types loop
    1. do-while
    2. while
    3. for
    4. foreach (it is also called iterator)
- If we want to manipulate elements of array then we should use java.util.Arrays class.

Day 5

- In java, checking array bounds is a job of JVM.
- Using illegal index, if we try to access element from array then JVM throws ArrayIndexOutOfBoundsException.

```
int[] arr = new int[ ] { 10, 20, 30 };
//int element = arr[ -1  ]; //ArrayIndexOutOfBoundsException
int element = arr[ arr.length  ];//ArrayIndexOutOfBoundsException
```

- foreach loop is also called as iterator in java.
- It is used to traverse collection in forward direction only. During travsering, we can only read the values.

Array of value type

```
boolean[] arr = new boolean[ 3 ];
```

- If we create array of value type then default value of array depends on default value of data type.

Array of references

```
//Single dynamic object in C++
Complex *ptr = new Complex( );
//Array of objects in C++
Complex *ptr = new Complex[ 3 ];
```

```
Complex c1;
//c1 is object reference / reference
Complex c1 = new Complex();
//It is instantiation of single java instance

Complex[] arr;
//arr is reference of single dimensional array

Complex[] arr = new Complex[ 3 ];
//It is array of references in java whose default value is null.
```

Array of instances

```
Complex c1 = new Complex();
Complex c2 = new Complex();
Complex c3 = new Complex();
```

```
Complex[] arr = new Complex[ 3 ];
arr[ 0 ] = new Complex();
arr[ 1 ] = new Complex();
arr[ 2 ] = new Complex();
```

```
Complex[] arr = new Complex[ 3 ];
for( int i = 0; i < arr.length; ++ i )
    arr[ i ] = new Complex();
```

Passing argument by reference:

- In java, we can pass argument to the method by value only.
- Using array, we can pass, argument to the method by reference.

```
private static void swap(int[] arr)
{
    int temp = arr[ 0 ];
    arr[ 0 ] = arr[ 1 ];
    arr[ 1 ] = temp;
}
public static void main(String[] args)
{
    int a = 10;
    int b = 20;

    int[] arr = new int[ ] { a, b };
    Program.swap( arr );
    a = arr[ 0 ]; b = arr[ 1 ];

    System.out.println("a       :       "+a);
    System.out.println("b       :       "+b);
}
```

Variable argument method

```
private static void sum( int... args )
{
    int result = 0;
    for( int element : args )
        result = result + element;
    System.out.println("Result  :       "+result);
}
```

- public static String format(String format, Object... args);
- public PrintStream printf(String format, Object... args);
- public Object invoke(Object obj, Object... args);

System Date and Time

- Using Calendar:
  - Calendar is abstract class declared in java.util package.
  - Fields:
    1. public static final int DATE
    2. public static final int MONTH
    3. public static final int YEAR
    4. public static final int HOUR
    5. public static final int MINUTE
    6. public static final int SECOND
  - Methods
    1. public static Calendar getInstance()
    2. public int get(int field)

```
Calendar c = Calendar.getInstance();
int day = c.get(Calendar.DATE);
int month = c.get(Calendar.MONTH) + 1;
int year = c.get(Calendar.YEAR);
```

- Using Date:
  - It is a concrete class declared in java.util package.
  - It is Deprecated class.
  - If we want to format Date and Time then we should use SimpleDateFormat class which is declared in java.text package.

```
//Date date = new Date(119, 10, 5);
Date date = new Date();
String pattern = "dd/MM/yyyy";
SimpleDateFormat sdf = new SimpleDateFormat(pattern);
String strDate = sdf.format(date);
```

- Using LocalDate:
  - It is a final class declared in java.time package.

```
LocalDate ld = LocalDate.now();
int day = ld.getDayOfMonth();
int month = ld.getMonthValue();
int year = ld.getYear();
```

Hierarchy

- It is major pillar of oops.
- level / order / ranking of abstraction is called hierarchy.
- Purpose of hierarchy is to achieve reusability.
- Type of hierarchy:
    1. has-a : Association
    2. is-a : Inheritance
    3. use-a : Dependancy
    4. creates-a: Instantiation

Association

- If "has-a" relationship exist between two types then we should use association.
- Example:
    1. Car has-a engine
    2. Room has-a wall
- If object/instance is part of/ component of another instance then it is called association.
- Composition and aggregation are specialized form of association.

```
class Date
{   };
class Address
{   };
class Person
{
    string name;     //Association
    Date dob;        //Association
    Address currAddress; //Association
};
Person p;
```

```
class Date
{   };
class Address
{   };
class Person
{
    string *name;     //Association
    Date *dob;        //Association
    Address *currAddress; //Association
public:
    Person( )
    {
        this->name = new string();
        this->dob = new Date();
```

```
            this->currAddress = new Address();
        }
    };
    Person *ptr = new Person();
```

```
class Date
{
    private int day;
    private int month;
    private int year;
}
class Address
{
    private String city;
    private String state;
    private int pincode;
}
class Person
{
    private String name;
    private Date dob;
    private Address currAddress;
    public Person( )
    {
        this.name = new String();
        this.dob = new Date();
        this.currAddress = new Address();
    }
}
Person p = new Person();
```

- In java, object/instance do not contain another instance directly. In other words, association do not represent physical containment.

Modularity

- It is major pillar of oops
- It is used to minimize module dependancy.
- In java we can achive it using .jar, .war, .ear file.
- jar file is reusable component of java i.e if we want to create reusable library then we should create jar file.
- if we want to create jar file then we should use jar tool.
- AssociationLib(Project) : .jar
    - org.sunbeam.dac.lib(Package)
        1. Date
        2. Address
        3. Person
- AssociationTest(Project)
    - org.sunbeam.dac.test(Package)

1. Program

- Steps to create jar file
  1. Create java project (AssociationLib) and define types inside it.
  2. Right click on java project -> Export -> java -> jar file -> Next -> select type -> choose location for jar file -> click on finish.
- Steps to use jar file / steps to add jar file in classpath / runtime classpath / buildpath.

Inheritance

- If "is-a" relationship exist between two types then we should use inheritance.
- Example:
  1. Car is vehicle
  2. Water is liquid
- Without modifying implemetation of existing class if we want to extend meaning of that class then we should use inheritance.
- In java parent class is called super class and child class is called sub class.
- If we want extend class or create sub class then we should use extends keyword.

```
class Person
{   }
class Employee extends Person
{   }
```

- Java do not support private and protected mode of inheritance. Hence default mode of inheritance is public.

- In java, class can extend only one class.

- Except constructor, all the members of super class (including private and static) inherit into sub class.

- all the fields of super class inherit into sub class but only non static field get space inside instance.

- If functionality of super class method is logically incomplete then we should redefine that method in sub class.

- Inside method of sub class, if we want to access members of super class then we should use super keyword.

- From any constructor of sub class, by default, super class's parameterless constructor gets called. If we want to call any constructor of super class from constructor of sub class then we should use super statement.

- super statement must be first statement inside constructor body.

- During inheritance, members of super class inherit into sub class. Hence sub class instance can be considered as super class instance.

- Since sub class instance can be considered as super class instance, we can use it in place of super class instance.

```
Person p = new Person();    //Ok
Person p = new Employee();  //Ok : Upcasting
```

- Members of sub class do not inherit into super class. Hence super class instance can not be considered as sub instance.
- Since super class instance can not be considered as sub class instance, we can not use it in place of sub class instance.

```
Employee emp = new Employee();  //Ok
Employee emp = new Person( );   //Not OK
```

Typing:

- It is minor pillar of oops.
- It is also called as polymorphism
- Ability of an object/instance to take multiple forms is called polymorphism
- In java we can achive polymorphism using
    1. Method overloading
    2. Method overriding
- Using polymorphism, we can reduce maintenance of system.

Upcasting

- We can convert reference of sub class into reference of super class. It is called upcasting.
- Upcasting is used to minimize object dependancy in the code.

```
Employee emp = new Employee("ABC",23,1672,35000);
//Person p = ( Person)emp;      //Upcasting
Person p = emp; //Upcasting
```

- In above code, emp has access to all the members of super class as well as sub class. But p has access to only members of super class.

Downcasting

- We can not convert, reference of super class into reference of sub class. It is called downcasting.
- In case of downcasting, explicit typecasting is mandatory.

```
Person p=new Employee("ABC",23,1672,35000);
p.showRecord();
Employee emp = (Employee) p;//Downcasting
emp.displayRecord();
```

```
Person p = null;
System.out.println(p);//null
Employee emp = (Employee) p;//Downcasting
System.out.println(emp);//null
```

- If downcasting fails then JVM throws ClassCastException.

```
Person p = new Person();
Employee emp = (Employee) p;//Downcasting
//Output : ClassCastException
```

Dynamic Method dispatch

- In java, all the methods are by default virtual.
- In case of upcasting, process of calling method of sub class using reference of super class is called dynamic method dispatch(in C++ : runtime polymorphism)

```
class Person
{
        public void printRecord( )
        {   }
}
class Employee extends Person
{
        public void printRecord( )
        {}
}
public class Program
{
        public static void main(String[] args)
        {
                Person p = new Employee();
                p.printRecord(); //DMD
        }
}
```

Method overriding

- Process of redefining method of super class inside sub class is called method overriding.
- Rules of method overriding

1. Access modifier in sub class method should be same or it should be wider.
2. Return type in sub class method should be same or it should be sub type.
3. Method name, number of parameters and type of parameters in sub class method must be same.
4. Checked exception list in sub class method should be same or it should be sub set.

Final method

- If implementation method is logically 100% complete then we should declare such method final.
- Final method inherit into sub class but we can not override it in sub class.
- In java, we can not override following methods:

1. constructor
2. private method
3. static method
4. final method

- Overridden method can be declared as final.
- Examples:
    1. getClass
    2. wait
    3. notify
    4. notifyAll

Abstract method

- If implementation method is logically 100% incomplete then we should declare such method abstract.

- abstract is keyword in java.

- abstract method do not contain body.

- if we delare method abstract then it is mandatory to declare class abstract.

- Without declaring method abstract, we can declare class abstract.

- We can not instantiate abstract class. But we can create reference of it.

- Example:

    1. java.lang.Number
    2. java.lang.Enum
    3. java.util.Calendar
    4. java.util.Dictionary
    5. java.io.InputStream
    6. java.io.OutputStream

- It is mandatory to override abstract method in sub class otherwise sub class can be considered as abstract.

- If we extend abstract class then either we should override method in sub class or we should declare sub class abstract.

Final class

- If implementation of the class is logically 100% complete then we should declare such class final.
- We can not extend final class.
- Example:

1. java.lang.System
2. java.lang.Math
3. java.lang.String, StringBuffer, StringBuilder
4. All Wrapper classes
5. java.util.Scanner

Sole Constructor

- Constructor of super class, that is designed to call from constructor of sub class only is called sole constructor.

```java
abstract class A
{
        private int num1;
        private int num2;
        public A( int num1, int num2 ) //Sole Constructor
        {
                this.num1 = num1;
                this.num2 = num2;
        }
        public void printRecord( )
        {
                System.out.println("Num1        :       "+this.num1);
                System.out.println("Num2        :       "+this.num2);
        }
}
class B extends A
{
        private int num3;
        public B( int num1, int num2, int num3 )
        {
                super( num1, num2 );
                this.num3 = num3;
        }
        public void printRecord( )
        {
                super.printRecord();
                System.out.println("Num3        :       "+this.num3);
        }
}
public class Program
{
        public static void main(String[] args)
        {
                B b = new B(10,20, 30 );
                b.printRecord();
        }
}
```

instanceof

- It is operator.
- At runtime, if we want to check inheritance then we should use instanceof operator.
- It returns boolean value.

```
private static void accept(Shape shape)
{
    if( shape instanceof Rectangle )
    {
        Rectangle rect = (Rectangle) shape;
    }
    else
    {
        Circle c = (Circle) shape;
    }
}
```

Day 6

- Override is annotation declared in java.lang package.
- It is introduced in jdk1.5
- It helps developer to override method in subclass.
- Annotation always begins with @.

```
class A
{
        public void print( double a )
        {
                System.out.println("Super class");
        }
}
class B extends A
{
        @Override
        public void print( double a )
        {
                System.out.println("Sub class");
        }
}
```

Equals method

- If we want to compare state of variable/instance of value type then we should use operator ==.

```
int num1 = 10;
int num2 = 10;
if( num1 == num2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Equal
```

- We can use operator == with variable of reference type.
- If we want compare state of references then we should use operator ==.

```
Employee emp1 = new Employee("Abc", 12, 25000);
Employee emp2 = new Employee("Abc", 12, 25000);
if( emp1 == emp2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Not Equal
```

- If we want to compare state of instances then we should use equals method.
- "equals" is non final method of java.lang.Object class.
- Syntax: public boolean equals(Object obj);
- If we do not override equals method in sub class then its super class's equals method gets call.
- Equals method of java.lang.Object do not compare state of instances. It compares state of refereces.

```
class Object
{
    public boolean equals(Object obj)
    {
        return (this == obj);
    }
}
```

- If we want to compare state of instances then we should ovveride equals method in sub class.

```
@Override
public boolean equals( Object obj )
{
    if( obj != null )
    {
        Employee other = (Employee) obj;
        if( this.empid == other.empid )
            return true;
    }
    return false;
}
```

- In java, primitive types are not classes.

```
int num1 = 10;
int num2 = 10;
if( num1.equals( num2 ) )   //Not OK
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Compiler Error
```

- We can not use equals method with variable/instance of value type.

Boxing & AutoBoxing

- Process of converting state of instance of value type into reference type is called boxing.

```
int number = 10;
String strNumber = String.valueOf( number );
```

```
int number = 10;
String strNumber = Integer.toString(number);
```

```
int number = 10;
Integer i = Integer.valueOf(number);
```

- If boxing is done implicitly then it is called auto-boxing.

```
int number = 10;
Object obj = number; //AutoBoxing
```

UnBoxing & AutoUnBoxing

- Process of converting state of instance of reference type into value type is called unboxing.

```
String str = "125";
int number = Integer.parseInt(str);
```

```
Integer n1 = new Integer(125);
int n2 = n1.intValue();
```

- If unboxing is done implicitly then it is called auto unboxing.

```
Integer n1 = new Integer(125);
int n2 = n1;
```

Generics

- In java, if we want to write generic code then we should use generics.
- Generic Code without generics

```
class Box
{
        private Object object;
        public Object getObject()
        {
                return object;
        }
        public void setObject(Object object)
        {
                this.object = object;
        }
}
```

```
Object obj = new String();//Upcasting : OK
String str = (String)obj;//Downcasting : OK
```

```
Object obj = new Date();//Upcasting : OK
Date dt = (Date)obj;//Downcasting : OK
```

```
Object obj = new Date();//Upcasting : OK
String str = (String)obj;//Downcasting
//ClassCastException
```

```
Box b1 = new Box();
b1.setObject( new Date( 119, 10, 6 ));
String str = (String) b1.getObject();
//Output : ClassCastException
```

- Using java.lang.Object class we can not write type safe generic code. If we want to write typesafe generic code then we should use generics.
- By passing, datatype / type as argument, we can write generic code in java. Hence parameterized type is called generics.
- Generic code using generics:

```
class Box<T> //T -> Type Parameter Name
{
        private T object;
        public T getObject()
        {
                return object;
        }
        public void setObject(T object)
        {
                this.object = object;
        }
}
public class Program
{
        public static void main1(String[] args)
        {
                Box<Date> b1 = new Box<Date>(); //Date -> Type Argument
                b1.setObject(new Date());
                Date date = b1.getObject();
        }
}
```

Commonly use type parameter names:

1. T : Type
2. N : Number
3. E : Element
4. K : Key
5. V : Value
6. U,S : Second Type Parameters

Type Inference:

- An ability of compiler to detect type of argument at compile time and use it as a type argument is called type inference.

```
Box<Date> b1 = new Box<Date>(); //OK
Box<Date> b2 = new Box<>(); //OK
```

Raw Type:

- If we instantiate generic/paramerized type without type argument then parameterized type is called Raw type.

```
Box b1 = new Box(); //Box -> Raw type
//Box<Object> b1 = new Box<>();
```

- If we want to instantiate parameterized type then type argument must be reference type.

```
Box<int> b1 = new Box();    //Not OK
Box<Integer> b1 = new Box();    // OK
```

Need of Wrapper class

1. If we want to convert String into numeric type.

2. If we want to store numeric values inside instance of parameterized type then type argument must be wrapper class

- It is possible to specify multiple type parameters for the class/interface.

```
class HashTable<K,V>
{
        private K key;
        private V value;
        public void put( K key, V value )
        {
                this.key = key;
                this.value = value;
        }
        public K getKey()
        {
                return key;
        }
        public V getValue()
        {
                return value;
        }
}
public class Program
{
        public static void main(String[] args)
        {
                HashTable<Integer,String> ht = new HashTable<>( );
                ht.put(1, "DAC");
                System.out.println("Key :       "+ht.getKey());
                System.out.println("Value       :       "+ht.getValue());

        }
}
```

Why Generics?

- It gives us stronger type checking at compile time. In other words, it helps us to write type safe code.
- It completly eliminates explict type casting
- It helps us to implement generic algorithm and data structure.

Bounded Type Parameter

- If we want to put restriction on type / datatype that can be used as type argument then we must specify bounded type parameter

```
class Box<T extends Number >
{   }
```

```
//T extends Number : Bounded type parameter

public class Program
{
        public static void main(String[] args)
        {
                Box<Number> b1 = new Box<>();//OK
                Box<Integer> b2 = new Box<>();//Ok
                Box<Double> b3 = new Box<>();//Ok
                Box<String> b4=new Box<>(); //Not OK
                Box<Date> b5 = new Box<>(); //Not Ok
        }
}
```

- Specifying bounded type parameter is a job of class implementor.

ArrayList

- It is resizable array.
- It is a part of collection framework

```
ArrayList<Integer> list = null;
list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);

for( Integer element : list )
{
    System.out.println(element);
}
```

- On the basis of diffrent type argument we can not overload method.

Wild card

- In java, '?' is called wild card, which represent unknown type.
- Types of wild card
    1. Unbounded wild card
    2. Upper bounded wild card
    3. Lower bounded wild card

Unbounded wild card

```
private static void print(ArrayList<?> list)
{
    for( Object element : list )
```

```
        System.out.println(element);
}
```

- In above code, list can contain refernce of ArrayList which can contain unknown type of element.

Upper bounded wild card

```
private static void print(
    ArrayList<? extends Number> list)
{
    for( Number element : list )
        System.out.println(element);
}
```

- In above code, list can contain reference of ArrayList, which can contain elements of Number or its sub type.

Lower bounded wild card

```
private static void print(
    ArrayList<? super Integer> list)
{
    for( Object element : list )
        System.out.println(element);
}
```

- In above code, list can contain reference of ArrayList which can contain elements of Integer and its super type.
- In type argument, we can not use inheritance.

```
private static void print(
    ArrayList<Integer> list)
{
    //TODO
}
ArrayList<Integer> intList = Program.getIntegerList( );

Program.print( intList ); //OK
```

```
private static void print(
    ArrayList<Number> list)
{
    //TODO
}
```

```
ArrayList<Integer> intList = Program.getIntegerList( );

Program.print( intList ); //Not OK
```

Generic Method

- generic method without generics:

```
public static void print( Object obj )
{
    System.out.println(obj.toString());
}
```

- generic method using generics:

```
public static <T> void print( T obj )
{
    System.out.println(obj.toString());
}
```

- Generic method with bounded type parameter

```
public static <T extends Number>
void print( T obj )
{
    System.out.println(obj.toString());
}
```

Restrictions on generics

- During instantation of parameterized type, type argument must be reference type.
- On the basis of only different type argument, we can not overload method.
- We can not instantiate type parameter

```
public static <T > void print( T obj )
{
    T t = new T(); //Not Ok
    //TODO
}
```

- we can not declare, parameterized type fields static.

```
class Box<T>
{
        private static T object;
}
```

- We can not use instanceof operator with parameterized type.

```
List<Integer> list = new ArrayList<>();
if( list instanceof ArrayList<Integer>)
//Not OK
{   }
```

Exception Handling

- Exception is an object/instance, which is used to send notification to the end user if exceptional situation occurs in the program.
- We should handle exception
  1. To manage runtime errors centrally(inside main method)
  2. To avoid resource leakage.
- Operating System Resources
  1. Memory
  2. File
  3. Thread
  4. Socket
  5. Nework Connection
  6. IO devices.
- If we want to handle exception then we should use five keywords:
  1. try
  2. catch
  3. throw
  4. throws
  5. finally
- AutoCloseable is interface declared in java.lang package.
- "void close() throws Exception" is a method of java.lang.AutoCloseable
- Closeable is interface declared in java.io package.
- "void close() throws IOException" is a method of java.io.Closeable interface.

Resource

- An instance, whose type implements AutoCloseable/Closeable interface is called resource.

```
class Test implements AutoCloseable
{
        @Override
        public void close() throws Exception
        {       }
}
class Program
{
    public static void main(String[] args)
    {
        Test t = new Test(); //resource
    }
}
```

Exception class hierarchy

- java.lang.Throwable is a super class of all errors and exceptions in java lanaguage.
- If runtime error gets generated due to runtime enviroment then it is considered as Error in context of exception handling.
- We can not recover from error.
- We can write try catch block to handle errors. But we can not recover from error hence it is not recommended to try try catch block to handle errors.
- Example:
    1. StackOverflowError
    2. VirtualMachineError
    3. OutOfMemoryError
- If runtime error gets generated due to application then it is considered as Exception in context of exception handling.
- We can recover from exception.
- Since it is possible to recover from exception, it is recommended to write try catch block to handle exception.
- Example:
    1. NullPointerException
    2. ClassCastException
    3. ClassNotFoundException

Types of exception

```
1. Checked Exception
2. Unchecked Exception
```

- Above types of exception are designed for java compiler.

Unchecked Exception

- java.lang.RuntimeException and all of its sub classes are considered as Unchecked exception.
- Handling unchecked exception is optional.
- Example:
    1. NumberFormatException
    2. NullPointerException
    3. NegativeArraySizeException
    4. ArrayIndexOutOfBoundsException
    5. ClassCastException

Checked Exception

- java.lang.Exception and all its sub classes except java.lang.RuntimeException(and its sub classes) are considered as checked exception.
- It is mandatory to handle checked exception.
- Example:
    1. CloneNotSupportedException
    2. InterruptedException
    3. ClassNotFoundException
    4. FileNotFoundException
    5.

Throwable

- It is a class declared in java.lang package.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM or can be thrown by the Java throw statement.
- Similarly, only Throwable class or one of its subclasses can be the argument type in a catch clause
- Constructor(s):

1. public Throwable()

```
Throwable t = new Throwable( );
```

2. public Throwable(String message)

```
Throwable t = new Throwable( "Exception" );
```

3. public Throwable(Throwable cause)

```
String msg = "Exception";
Throwable cause = new Throwable( msg);
Throwable t = new Throwable( cause);
```

4. public Throwable(String message, Throwable cause)

```
String msg = "Exception";
Throwable cause = new Throwable();
Throwable t = new Throwable( msg, cause);
```

- Method(s)

1. public String getMessage()
2. public Throwable getCause()
3. public void printStackTrace()

try

- It is keyword in java
- It is used to inspect exception.
- In java, try block must have at least one catch block, finally block or resource.

catch

- It is keyword in java
- It is used to handle exception.
- For single try block we can provide multiple catch block.
- In single catch block, we can handle multiple specific exceptions. such catch block is called multi catch block.

```
try
{       }
catch( ArithmeticException | InputMismatchException ex )
{
        //TODO
}
```

- NullPointerException is a unchecked exception.

```
NullPointerException ex = new NullPointerException();   //OK

RuntimeException ex = new NullPointerException();       //OK
```

```
Exception ex = new NullPointerException();//OK
```

- Interrupted Exception is a checked exception.

```
Interrupted ex = new InterruptedException();//OK
Exception ex = new InterruptedException();//OK
```

- java.lang.Exception class reference variable can contain reference of any checked as well as unchecked exception. Hence to write generic catch block we should use Exception class.
- Syntax:

```
try
{
        //TODO
}
catch( Exception ex )//Generic catch block
{
        ex.printStackTrace();
}
```

- If child/parent relation is exist between exception types then we must handle child type exceptions first.

```
try
{
        //TODO
}
catch (ArithmeticException ex)
{       }
catch (RuntimeException ex)
{       }
catch (Exception ex)
{       }
```

throw

- It is keyword in java.
- It is used to generate new exception
- using throw keyword, we can throw instance of sub class of java.lang.Throwable class only.
- throw statement is jump statement.

```
try
{
        System.out.print("Num1  :       ");
        int num1 = sc.nextInt();
        System.out.print("Num2  :       ");
        int num2 = sc.nextInt();
        if( num2 == 0 )
                throw new ArithmeticException("Divide by zero exception");
        int result = num1 / num2;
        System.out.println("Result      :       "+result);
}
catch (ArithmeticException ex)
{
        System.out.println(ex.getMessage());
}
```

finally

- It is keyword in java.
- If we want to release local resources then we should use finally block.
- JVM always execute finally block.
- for try block we can provide only one finally block.
- If we write System.exit(0) inside try and catch block then JVM do not execute finally block.

Day 7

Exception Handling

try with resource

```
try( Scanner sc = new Scanner(System.in))
{
    System.out.print("Number    :       ");
    int number = sc.nextInt();
    System.out.println("Number  :       "+number);
}
```

- If we use resource with try then it is not necessary to write finally block. In this case, JVM implicitly gives call to the close function.

throws clause

- If we want to delegate exception(checked/unchecked) from one method to another method then we should use throws clause.

```
public static void printRecord( ) throws InterruptedException
{
    for( int count = 1; count <= 10; ++ count )
    {
        System.out.println("Count: "+count);
        Thread.sleep(250);
    }
}
public static void main(String[] args)
{
    try
    {
        Program.printRecord();
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
```

Custom Exception

- JVM can understand exceptional conditions that is occurred in business logic. If we want to handle such situations then we should write custom exception class.

- If we want to define custom unchecked exception class then we should extend the class from java.lang.RuntimeException class.

```
class StackOverflowException extends RuntimeException
{   }
```

- If we want to define custom checked exception class then we should extend the class from java.lang.Exception class.

```
class StackOverflowException extends Exception
{   }
```

Exception Chaining

- Generally exceptions are handled by throwing new type of exception. It is called exception chaining
- If we want trace any application then we should use exception chaining.

Bug

- If runtime error gets generated due to application developer's mistake then it is considered as bug.
- Example:
    1. NullPointerException
    2. ArrayIndexOutOfBoundsException
    3. ClassCastException
- We should not provide try catch block to handle bug rather we should find out cause of the bug.

Exception

- If runtime error gets generated due to end users mistake then it is considered as Exception.
- Example:
    1. ClassNotFoundException
    2. FileNotFoundException
- We should provide try catch block to handle exception

Error

- If runtime error gets generated due to enviromental condition then it is considered as error.

Fragile Base Class Problem

- If we make changes in the method of super class then it is necessary to recompile that class and all its sub classes. It is called fragile base class problem.

Interface

- Set of rules is called specification/standard.

- It is a java language feature that is used to define specifications for the sub classes.
- Interface help us:
    1. to develop/build trust realtionship between service provider and service consumer.
    2. to minimize vendor dependancy(to achive loose coupling)
- It is reference type.
- interface is keyword in java.
- Interface can contain:
    1. Nested interface
    2. Constants(Final Fields)
    3. Abstract Method
    4. Default Method
    5. Static Method
- We can declare fields inside interface. Interface fields are implicitly considered as public static and final.

```
interface A
{
        int number = 10;
        //public static final int number = 10;
}
```

- Interface methods are by default considered as public and abstract.

```
interface A
{
    void print();
        //public abstract void print();
}
```

- We can not define constructor inside interface.
- We can not instantiate interface but we can create reference of interface.
- Using "implements" keyword, we can define rules in sub class.

```
interface A
{
        int number = 10;
        //public static final int number = 10;

        void print();
        //public abstract void print();
}
class B implements A
{
        @Override
        public void print()
        {
                System.out.println("Inside B.print");
        }
}
public class Program
{
        public static void main(String[] args)
        {
                A a = new B();
                a.print();
        }
}
```

- It is mandatory to override abstract methods of interface otherwise sub class can be considered as abstract.

Interface Inheritance:

- In inheritance, if super type and sub type is interface then it is called interface inheritance.

Interface Implementation Inheritance:

- In inheritance, if super type is interface and sub type is class then it is called interface Implementation inheritance.

Implementation Inheritance

- In inheritance, if super type and sub type is class then it is called implementation inheritance.

- I1, I2, I3 -> Interfaces

- C1, C2, C3 -> Classes

1. I2 implements I1; //Not OK
2. I2 extends I1; // OK
3. I3 extends I1, I2; // OK
4. I1 extends C1; //Not OK
5. C1 extends I1; //Not OK
6. C1 implements I1; //OK
7. C1 implements I1, I2; //OK
8. C2 implements C1; //Not OK
9. C2 extends C1; //OK
10. C3 extends C1, C2; //Not OK
11. C2 extends C1 implements I1, I2; //OK

- If interfaces having method with same name then sub class can override it only once.

When to use abstract class and interface?

- If "is-a" realationship exist between super type & sub type and if we want to provide same method design in all the sub classes then we should declare super type abstract.
- Using abstract class, we can group objects/instances of related types together.

```
Shape[] arr = new Shape[ 3 ];
arr[ 0 ] = new Rectangle();
arr[ 1 ] = new Circle();
arr[ 2 ] = new Triangle();
```

- Abstract class can extends only one class(abstract/concrete).
- We can write constructor inside abstract class.
- Abstract class may/may not contain abstract method.
- If "is-a" realationship is not exist between super type & sub type and if we want to provide same method design in all the sub classes then we should declare super type interface.
- Using interface, we can group instances of unrelated type together.

```
Printable[] arr = new Printable[ 3 ];
arr[ 0 ] = new Complex();
arr[ 1 ] = new Date();
arr[ 2 ] = new Point();
```

- Interface can extends more than one interfaces.
- We can not define constructor inside interface.
- Interface methods are by default abstract.

Adapter class

- Abstract helper class, which allows us to override some of the methods of interface is called Adapter class.

```
interface A
{
        void f1();
        void f2();
        void f3();
}
abstract class B implements A //Adpater class
{
        @Override
        public void f1() {      }
        @Override
        public void f2() {      }
        @Override
        public void f3() {      }
}
```

Cloneable Implementation

- If we want to create new instance from existing instance then we should use clone() method.
- clone() is non final method of java.lang.Object class.
- Syntax: protected native Object clone() throws CloneNotSupportedException
- Inside clone() method, if we want to create shallow copy of current instance then we should use "super.clone()"
- Cloneable is marker interface declared in java.lang package.
- Without implementing, Cloneable interface, if we try to create clone of instance then clone() methods throws CloneNotSupportedException.

Marker Interface

- An interface which do not contain any member is called marker interface.
- It is also called as tagging interface.
- Main purpose of marker interface is to generate metadata for JVM.
- Example:
    1. java.lang.Cloneable
    2. java.util.EventListener
    3. java.util.RandomAccess
    4. java.io.Serializable
    5. java.rmi.Remote

Iterable & Iterator implementation

- Iterable is interface declared in java.lang package.
- "Iterator iterator()" is abstract method of java.lang.Iterable interface.
- It is a factory method for Iterator.
- Iterator is interface declared in java.util package.
- Abstract methods of java.util.Iterator interface:
    1. boolean hasNext()
    2. E next()
- Using foreach loop we can traverse elements of array and instance whose type implements java.lang.Iterable interface.
- If class implements Iterable interface then it is considered as travseribile for "for-each" loop.
- In C++, if we use any instance as a pointer then it is called smart pointer.
- Iterator is smart pointer that is used to traverse collection.

Day 8

Comparable and Comparator implementation

- If we want to sort array of value type using Arrays.sort() then sort() method implicitly use "Dual-Pivot Quicksort" algorithm.
- Comparable is interface declared in java.lang package.
- "int compareTo(T other)" is a method of Comparable interface.
- If we want to sort array of instances of same type then reference type must implement Comparable interface.
- compareTo() method returns integer value:
  - Returns a negative integer, zero, or a positive integer as current object is less than, equal to, or greater than the specified object.
- If we use Arrays.sort() method to sort array of instances of reference type then sort() method implicitly use "iterative mergesort" algorithm.
- Comparator is interface declared in java.util package.
- "int compare(T o1,T o2)" is a method of Comparator interface.
- If we want to sort array of instances of same type as well different type then we should use Comparator interface.
- compare method returns integer value.
  - Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
- If any class implements Comparable interface then it is considered as sortable.
- All the wrapper classes implements Comparable interface.

Collection Framework

- Every value/data stored in data structure is called element.
- In java, data structure class is called collection.
- Framework is library of reusable classes/interafces that is used to develop applicaiton.
- Library of reusable data structure classes that is used to develop java application is called collection framework.
- Main purpose of collection framework is to manage data in RAM efficiently.
- Consider following Example:
  1. Person has-a birthdate
  2. Employee is a person
- In java, collection instance do not contain instances rather it contains reference of instances.
- If we want to use collection framework them we should import java.util package.

Iterable:

- It is a interface delared in java.lang package.
- All the collcetion classes implements Iterable interface hence we can traverse it using for each loop
- Methods of Iterable interface

1. Iterator iterator()

2. default Spliterator spliterator()
3. default void forEach(Consumer<? super T> action)

Collection

- Collection is interface declared in java.util package.
- It is sub interface of Iterable interface.
- It is root interface in collection framework interface hierarchy.
- Abstract Methods of Collection Interface

1. boolean add(E e)
2. boolean addAll(Collection<? extends E> c)
3. void clear()
4. boolean contains(Object o)
5. boolean containsAll(Collection<?> c)
6. boolean isEmpty()
7. boolean remove(Object o)
8. boolean removeAll(Collection<?> c)
9. boolean retainAll(Collection<?> c)
10. int size()
11. Object[] toArray()
12. T[] toArray(T[] a)

- Default methods of Collection interface

1. default Stream stream()
2. default Stream parallelStream()
3. default boolean removeIf(Predicate<? super E> filter)

List

- It is sub interface of java.util.Collection interface.
- It is ordered/sequential collection.
- ArrayList, Vector, Stack, LinkedList etc. implements List interface. It generally refered as "List collections".
- List collection can contain duplicate element as well multiple null elements.
- Using integer index, we can access elements from List collection.
- We can traverse elements of List collection using Iterator as well as ListIterator.
- It is introduced in jdk 1.2.
- Note: If we want to manage elements of non final type inside List collection then non final type should override "equals" method.
- Abstract methods of List Interface

1. void add(int index, E element)
2. boolean addAll(int index, Collection<? extends E> c)
3. E get(int index)
4. int indexOf(Object o)
5. int lastIndexOf(Object o)
6. ListIterator listIterator()
7. ListIterator listIterator(int index)

8. E remove(int index)
9. E set(int index, E element)
10. List subList(int fromIndex, int toIndex)

- Default methods of List interface

1. default void sort(Comparator<? super E> c)
2. default void replaceAll(UnaryOperator operator)

ArrayList

- It is resizable array.
- It implements List, RandomAccess, Cloneable, Serializable interfaces.
- It is List collection.
- It is unsynchronized collection. Using "Collections.synchronizedList" method, we can make it synchronized.

```
List list = Collections.synchronizedList(new ArrayList(...));
```

- Initial capacity of ArrayList is 10. If ArrayList is full then its capacity gets increased by half of its existing capacity.
- It is introduced in jdk 1.2
- Note: If we want to manage elements of non final type inside ArrayList then non final type should override "equals" method.
- Constructor(s) of ArrayList

1. public ArrayList()

```
ArrayList<Integer> list = new ArrayList<>();
List<Integer> list = new ArrayList<>();
Collection<Integer> list = new ArrayList<>()
```

2. public ArrayList(int initialCapacity)

```
ArrayList<Integer> list =
              new ArrayList<>(15);
List<Integer> list = new ArrayList<>(15);
Collection<Integer> list =
                new ArrayList<>(15)
```

3. public ArrayList(Collection<? extends E> c)

```
Collection<Integer> c = ArrayList<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = Vector<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = TreeSet<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = ArrayDeque<>();
List<Integer> list = new ArrayList<>(c);
```

- Methods of ArrayList

1. public void ensureCapacity(int minCapacity)
2. protected void removeRange(int fromIndex, int toIndex)
3. public void trimToSize()

- Using illegal index, if we try to access element from any List collection then List methods throws IndexOutOfBounds Exception.

```
List<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);
Integer element = list.get(list.size());
//Output : IndexOutOfBoundsException
```

- If we want to sort elements of array then we should use Arrays.sort() method and to sort elements of List collection, we should use Collections.sort() method.

Vector

- It is resizable array.
- It implements List, RandomAccess, Cloneable, Serializable.
- It is List collection.
- It is synchronized collection.
- Default capacity of vector is 10. If vector is full then its capacity gets increased by its existing capacity.
- We can traverse elements of vector using Iterator, ListIterator as well as Enumeration.
- It is introduced in jdk 1.0.
- Note: If we want to manage elements of non final type inside Vector then non final type should override "equals" method.

Following classes are by default synchronized

1. Vector
2. Stack(Sub class of Vector)
3. Hashtable
4. Properties(Sub class of Hashtable)

Enumeration

- It is interface declared in java.util package.
- Methods of Enumeration I/F
    1. boolean hasMoreElements()
    2. E nextElement()
- It is used to traverse collection only in forward direction. During traversing, we can add, set or remove element from collection.
- It is introduced in jdk 1.0.
- "public Enumeration elements()" is a method of Vector class.

```
Vector<Integer> v = new Vector<Integer>();
v.add(10);
v.add(20);
v.add(30);

Integer element = null;
Enumeration<Integer> e = v.elements();
while( e.hasMoreElements())
{
    element = e.nextElement();
    System.out.println(element);
}
```

Iterator

- It is a interface declared in java.util package.
- It is used to traverse collection only in forward direction. During traversing, we can not add or set element but we can remove element from collection.
- Methods of Iterator

1. boolean hasNext()
2. E next()
3. default void remove()
4. default void forEachRemaining(Consumer<? super E> action)

- It is introduced in jdk 1.2

```
Vector<Integer> v = new Vector<Integer>();
v.add(10);
v.add(20);
v.add(30);
```

```
Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.  println(element);
}
```

ListIterator

- It is subinterface of Iterator interface.
- It is used to traverse only List Collection in bidirection.
- During traversing we can add, set as well as remove element from collection.
- It is introduced in jdk 1.2
- Methods of ListIterator

1. boolean hasNext()
2. E next()
3. boolean hasPrevious()
4. E previous()
5. void add(E e)
6. void set(E e)
7. void remove()

```
Vector<Integer> v = new Vector<Integer>();
v.add(10);
v.add(20);
v.add(30);

Integer element = null;
    ListIterator<Integer> itr = v.listIterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.print(element+"  ");
}
System.out.println();
while( itr.hasPrevious())
{
    element = itr.previous();
    System.out.print(element+"  ");
}
```

Day 9

Collection Framework

Types of Iterator

1. Fail Fast Iterator
2. Fail Safe Iterator(Not Fail Fast)

Fail Fast Iterator

- During traversing, using collection reference, if we try to modify state of collection and if iterator do not allows us to do the same then such iterator is called "Fail Fast" Iterator. In this case JVM throws ConcurrentModificationException.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.print(element+"  ");
    if( element == 50 )
        v.add(60); //ConcurrentModificationException
}
```

Fail safe Iterator

- During traversing, if itrator allows us to do changes in underlying collection then such iterator is called fail safe iterator.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Enumeration<Integer> e = v.elements();
```

```
while( e.hasMoreElements())
{
    element = e.nextElement();
    System.out.print(element+" ");
    if( element == 50 )
        v.add(60); //OK
}
```

Stack

- It is linear data structure which is used to manage elements in Last In First Out order.
- It is sub class of Vector class.
- It is synchronized collection.
- It is List Collection.
- Methods of Stack class
    1. public boolean empty()
    2. public E push(E item)
    3. public E peek()
    4. public E pop()
    5. public int search(Object o)

```
Stack<Integer> stk = new Stack<Integer>();
stk.push(10);
stk.push(20);
stk.push(30);
Integer element = null;
while( !stk.empty() )
{
    //element = stk.peek();
    element = stk.pop();
    System.out.println("Popped element is : "+element);
}
```

- Since it is synchronized collection, it slower in performance.
- For high performance we should use ArrayDeque class.

```
Deque<Integer> stk = new ArrayDeque<>();
stk.push(10);
stk.push(20);
stk.push(30);
Integer element = null;
while( !stk.isEmpty())
{
    element = stk.peek();
    System.out.println("Popped element is : "+element);
    stk.pop();
}
```

LinkedList

- It is a List collection.
- It implements List, Deque, Cloneable and Serializable interface.
- Its implementation is depends on Doubly linked list.
- It is unsynchronized collection. Using Collections.synchronizedList() method, we can make it synchronized.

```
List list = Collections.synchronizedList(new LinkedList(...));
```

- It is introduced in jdk 1.2.
- Note : If we want to manage elements of non-final type inside LinkedList then non final type should override "equals" method.
- Instantiation

```
List<Integer> list = new LinkedList<>();
```

Queue

- It is interface declared in java.util package.
- It is sub interface of Collection interface.
- It is introduced in jdk 1.5
- Option 1

```
Queue<Integer> que = new ArrayDeque<>();
que.add(10);
que.add(20);
que.add(30);
Integer ele = null;
while( !que.isEmpty())
{
    ele = que.element();
    System.out.println("Removed element is : "+ele);
    que.remove();
}
```

- Option 2

```
public static void main(String[] args)
{
    Queue<Integer> que = new ArrayDeque<>();
    que.offer(10);
    que.offer(20);
    que.offer(30);
```

```
        Integer ele = null;
        while( !que.isEmpty())
        {
            ele = que.peek();
            System.out.println("Removed element is : "+ele);
            que.poll();
        }
    }
```

Deque

- It is usually pronounced "deck".
- It is sub interface of Queue.
- It is introduced in jdk 1.6
- If we want to perform operations from bidirection then we should use Deque interface.

Set

- It is sub interface of java.util.Collection interface.
- HashSet, LinkedHashSet, TreeSet etc. implements Set interface. It is also called as Set collection.
- Set collections do not contain duplicate elements.
- It is introduced in jdk 1.2

TreeSet

- It is Set collection.
- It can not contain duplicate element as well as null element.
- It is sorted collection.
- Its implementation is based on TreeMap
- It is unsynchronized collection. Using "Collections.synchronizedSortedSet()" method we can make it synchronized.

```
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
```

- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside TreeSet then non final type should implement Comparable interface.
- Instantiation

```
Set<Integer> set = new TreeSet<>( );
```

Searching

- It is process of finding location(index/address/reference) of element inside collection.

- Commonly used searching techniques are:
    1. Linear / Sequential Search
    2. Binary Search
    3. Hashing
- In array, elements are stored sequentially. Hence time required to earch every element is different.
- Hashing is a searching algorithm which is used to search element in constant time(faster searching).
- In case array, if we know index of element then we can locate it very fast.
- Hashing technique is based on "hashcode".
- Hashcode is not a reference or address of the object rather it is a logical integer number that can be generated by processing state of the object.
- Generating hashcode is a job of hash function/method.
- Generally hashcode is generated using prime number.

```
//Hash Method
private static int getHashCode(int data)
{
    int result = 1;
    final int PRIME = 31;
    result = result * data + PRIME * data;
    return result;
}
```

- If state of object/instance is same then we will get same hashcode.
- In hashing, index is called slot.
- Hashcode is required to generate slot.
- If state of objects are same then their hashcode and slot will be same.
- By processing state of two different object's , if we get same slot then it is called collision.
- Collision resolution techniques:
    1. Seperate Chaining / Open Hashing
    2. Open Addressing / Close Hashing
        1. Linear Probing
        2. Quadratic Probing
        3. Double Hashing / Rehashing
- Collection(LinkedList/Tree) maintained per slot is called bucket.
- Load Factor = (Count of bucket / Total elements);
- In hashcode based collection, if we want manage elements of non final type then refernce type should override equals() and hashcode() method.
- hashCode() is non final method of java.lang.Object class.
- Syntax: public native int hashCode( );
- On the basis of state of the object, we want to generated hashcode then we should ovveride hashCode() method in sub class.

HashSet

- It Set Collection.
- It can not contain duplicate elements but it can contain null element.

- It's implementation is based on HashTable.
- It is unordered collection.
- It is unsynchronized collection. Using Collections.synchronizedSet() method, we can make it synchronized.
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside HashSet then non final type should override equals and hashCode() method.
- Instantiation:

```
Set<Integer> set = new HashSet<>();
```

LinkedHashSet

- It is sub class of HashSet class.
- Its implementation is based on linked list and Hashtable.
- It is ordered collection.
- It is unsynchronized collection. Using Collections.synchronizedSet() method we can make it synchronized.

```
Set s = Collections.synchronizedSet(new LinkedHashSet(...));
```

- It is introduced in jdk 1.4
- It can not contain duplicate element but it can contain null element.

Dictionary<K,V>

- It is abstract class declared in java.util package.
- It is super class of Hashtable.
- It is used to store data in key/value pair format.
- It is not a part of collection framework
- It is introduced in jdk 1.0
- Methods:

1. public abstract boolean isEmpty()
2. public abstract V put(K key, V value)
3. public abstract int size()
4. public abstract V get(Object key)
5. public abstract V remove(Object key)
6. public abstract Enumeration keys()
7. public abstract Enumeration elements()

- Implementation of Dictionary is Obsolete.

Map<K,V>

- It is part of collection framework but it doesn't extend Collection interface.

- This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.
- HashMap, Hashtable, TreeMap etc are Map collection's.
- Map collection stores data in key/value pair format.
- In map we can not insert duplicate keys but we can insert duplicate values.
- It is introduced in jdk 1.2
- Map.Entry<K,V> is nested interface of Map<K,V>.
- Following are abstract methods of Map.Entry interface.
    1. K getKey()
    2. V getValue()
    3. V setValue(V value)
- Abstract Methods of Map<K,V>

1. boolean isEmpty()
2. V put(K key, V value)
3. void putAll(Map<? extends K,? extends V> m)
4. int size()
5. boolean containsKey(Object key)
6. boolean containsValue(Object value)
7. V get(Object key)
8. V remove(Object key)
9. void clear()
10. Set keySet()
11. Collection values()
12. Set<Map.Entry<K,V>> entrySet()

- An instance, whose type implements Map.Entry<K,V> interface is called enrty instance.

```
class Pair<K,V> implements Entry<K, V>
{
        private K key;
        private V value;
        @Override
        public K getKey()
        {
                return this.key;
        }
        @Override
        public V getValue()
        {
                return this.value;
        }
        @Override
        public V setValue(V value)
        {
                this.value = value;
                return this.value;
        }
}
```

```
class Program
{
    public static void main(String[] args)
    {
        Entry<Integer, String> e = new Pair<>();
        //Here pair instance is entry instance
    }
}
```

- Map is collection of entries where each entry contains key/value pair.

Hashtable

- It is Map<K,V> collection which extends Dictionary class.
- It can not contain duplicate keys but it can contain duplicate values.
- In hashtable, Key and value can not be null.
- It is synchronized collection.
- It is introduced in jdk 1.0
- In Hashtable, if we want to use instance non final type as key then it should override equals and hashCode method.

Day 10

HashMap<K,V>

- It is map collection
- It's implementation is based on Hashtable.
- It can not contain duplicate keys but it can contain duplicate values.
- In HashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method, we can make it synchronized.

```
Map m = Collections.synchronizedMap(new HashMap(...));
```

- It is introduced in jdk 1.2.
- Instantiation

```
Map<Integer, String> map = new HashMap<>( );
```

- Note : In HashMap, if we want to use element of non final type as a key then it should override equals() and hashCode() method.

LinkedHashMap<K,V>

- It is sub class of HashMap<K,V> class
- Its implementation is based on LinkedList and Hashtable.
- It is Map collection hence it can not contain duplicate keys but it can contain duplicate values.
- In LinkedHashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method we can make it synchronized.

```
Map m = Collections.synchronizedMap(new LinkedHashMap(...));
```

- LinkedHashMap maintains order of entries according to the key.
- Instantiation:

```
Map<Integer, String> map = new LinkedHashMap<>();
```

- It is introduced in jdk 1.4

TreeMap

- It is map collection.
- It can not contain duplicate keys but it can contain duplicate values.
- It TreeMap, key not be null but value can be null.
- Implementation of TreeMap is based on Red-Black Tree.
- It maintains entries in sorted form according to the key.
- It is unsynchronized collection. Using Collections.synchronizedSortedMap() method, we can make it synchronized.

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

- Instantiation:

```
Map<Integer, String> map = new TreeMap<>();
```

- It is introduced in jdk 1.2
- Note : In TreeMap, if we want to use element of non final type as a key then it should implement Comparable interface.

JDBC

- Java DataBase Connectivity
- Specification = { Abstract classes + Interfaces }
- JDBC is a specification defined by SUN/ORACLE.
- Implementating JDBC specification is a job of database vendor.
- Database connector(.jar) contains implementation of JDBC specification.
- e.g
    - mysql-connector-java-8.0.15.jar is connector for mysql database version 8.0.15.
- Using JDBC, we can access data from any RDBMS into java application.
- Main purpose of JDBC is to minimize database vendor dependancy.
- If we want use JDBC then we should import java.sql package.

JDBC Versions

- Since 1.8 : JDBC 4.2 API
- Since 1.7 : JDBC 4.1 API
- Since 1.6 : JDBC 4.0 API
- Since 1.4 : JDBC 3.0 API
- Since 1.2 : JDBC 2.0 API

JDBC Interfaces

1. Driver
2. Connection
3. Statement
4. PreparedStatement

5. CallableStatement
6. ResultSet
7. Blob
8. Clob
9. NClob
10. DatabaseMetaData
11. ResultSetMetaData
12. ParameterMetaData

JDBC Classes

1. DriverManager
2. Date
3. Time
4. Timestamp
5. Types

JDBC Exception

1. SQLException
2. SQLIntegrityConstraintViolationException

JDBC Driver:

- JDBC driver is a program, which converts Java request into SQL request and SQL response into Java response.
- All JDBC drivers must implement java.sql.Driver interface.
- Types of JDBC Driver:
    1. Type I
    2. Type II
    3. Type III
    4. Type IV

Type I Driver

- It is also called JDBC-ODBC bridge driver
- sun.jdbc.odbc.JdbcOdbcDriver
- Open DataBase Connectivity(ODBC)
- ODBC is specification defined by Microsoft to access data from databases.
- Type-I driver implicitly use ODBC driver.
- It comes with JDK.
- Vendor : Sun Microsystems
- It is database independant but platform dependant driver.

Advantages:

1. Since it comes with JDK, seperate installation is not required.

2. Easy to use.(Simple we need to configure System/User DSN).
3. It is database independant driver.

Disadvantages:

1. Since multiple conversions are involed, it is slower int performance.
2. Since it is based on ODBC, it is platform dependant(Mircrosoft specific).
3. Since jdk1.8 it is obsolete.

Type II Driver

- It is also called Native API Driver.
- e.g : Oracle OCI(Oracle Call Interface) Driver.
- In context of java, C/C++ code is called native code.
- If we want to use native code into Java code then we should use JNI.
- Type -II driver is based on JNI.
- Type-II driver do not use ODBC driver rather it uses vendor provided database specific native library.

Advantages:

1. It is faster than Type-I driver
2. It is more portable than Type-I driver

Disadvantages:

1. It is database dependant as well as platform dependant driver
2. Not all the vendors provide native library hence its use is limited.

Type III Driver

- It is also called as Network protocol driver or middleware driver.
- It follows n-tier architecture.
- e.g RMI Web Logic Driber
- It is database independant as well as platform independant driver.
- It is pure java driver which requires network setup.

Advantages:

1. Since it is written in java, it is truly portable.
2. Since it is database independant driver, we can use it to communicate with multiple databases
3. No need to use ODBC driver and vendor provided database specific native API lib.

Disadvantages:

1. It requires seperate network setup. Hence expensive to use.

Type IV Driver

- It is also called as Database protocol driver / thin driver / pure java driver.
- Since it is written in java it is truly portable.
- It is database dependant driver.
- We can use it with CUI/GUI as well as web application
- Either database vendor / third party can provide it.
- It implicitly use socket programming.
- e.g : com.mysql.jdbc.Driver

Advantages:

1. It is portable driver
2. Since implicitly use socket programming, no need to use odbc driver or database specific native lib
3. Easily available
4. Installation is easy.
5. We can use it with CUI as well as web application

Disadvantages:

1. It is database dependant driver

Steps to connect java application to the database.

- Step 0 : Add database connector(.jar) in buildpath
- Step 1 : Load and Register driver
- Step 2 : Establish Connection Using Users Credential
- Step 3 : Create Statement/PreparedStatement/CallableStatement object to execute query.
- Step 4 : Prepare and execute Query.
- Step 5 : Close resources

Configuration Information

- Database Server : MySQL 8.0.17
- Database/Schema : dac_db
- User Name : root
- Password : mysql@8017
- Driver : com.mysql.cj.jdbc.Driver
- URL : jdbc:mysql://localhost:3306/dac_db;
- To execute DML statements/queries (Insert/Update/Delete) we should use executeUpdate() method and to execute DQL statement(select) we should use executeQuery() method.
- executeUpdate() and executeQuery() are methods of java.sql.Statement interface.

```
Statement stmt = con.createStatement();
```

ResultSet

- It is interface declared in java.sql package.
- "ResultSet executeQuery(String sql) throws SQLException " is a method of Statement interface which returns ResultSet.
- ResultSet implementation object contains records returned by select query.
- A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next() method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.
- If we want to access data from ResultObject then we should use getXXX() method.
- To access data either we should use column index or column lable(name).

Object Relational Mapping(ORM)

- In SQL, Table is also called as Relation
- SQL terminologies
    1. Database
    2. Table
    3. Column
    4. Row
    5. Primary key / Foreign key
- OO Terminologies
    1. Class
    2. Field
    3. Instance / Object
- ORM Database Java Table Class Column Field Record Instance Primary key Identity Field
- If we want to map database enities with oo enity then we should define POJO class.

Plain Old Java Object(POJO)

- It is a class which do not implement interface or do not extend any class.
- It is also called as Data Transfer Object(DTO) / Business Object(BO) / Value Object(VO) / Entity.
- Rules to define POJO class

1. It must be packaged public class
2. It should contain default constructor
3. For every column it should contain private field inside class.
4. For every private field, it should contain getter and setter methods.
5. It is should not contain business logic but it can contain toString(), equals() or hashCode() method.

Data Access Object (DAO) Layer

- If we want to seperate data manipulation logic from business logic then we should define DAO class.
- Per table we should define one POJO and DAO.
- Rules to define DAO class.

1. It must be packaged public class.
2. It should contain default constructor
3. It should contain data manipulation logic(INSERT,UPDATE,DELETE, SELECT statement)

Day 11

Driver

- It is interface declared in java.sql package.
- JDBC Driver must implement java.util.Driver interface.
- It is used to establish connection between java application and database.
- If we want to use JDBC driver then first it must be registered.
- Registration can be done using 2 ways

First Way

```
Driver driver = ...;
DriverManager.registerDriver( driver );
```

Second Way

```
String driverName = "...";
Class.forName( driverName );
```

- Since Java SE 8 this step is optional.
- The DriverManager will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

DriverManager

- It is a class declared in java.sql package.
- DriverManager is responsible for locating proper driver and establishing connection with database using driver object.
- getConnection( ) is a method of DriverManager which returns Connection object.

```
Connection c = DriverManager.getConnection( url, user, password );
```

Connection

- It is interface declared in java.sql package.
- It represents session between java application and database.
- In other words, we can perform operations on database after establising connection and before closing connection.
- Methods of Connection Interface:

1. Statement createStatement() throws SQLException

2. PreparedStatement prepareStatement(String sql) throws SQLException
3. CallableStatement prepareCall(String sql) throws SQLException
4. void setAutoCommit(boolean autoCommit) throws SQLException
5. void commit() throws SQLException
6. void rollback() throws SQLException
7. Savepoint setSavepoint() throws SQLException

Statement

- It is interface declared in java.sql package.
- Instantiation :

```
Statement stmt = con.createStatement();
```

- If we want to execute static queries then we should use Statement object
- Limitations :
    1. It can not handle special characters in query
    2. It can not avoid SQL Injection.
    3. If use statement object to execute sql statement then query gets compiled and executed per request.
- Methods:
    1. ResultSet executeQuery(String sql) throws SQLException
    2. int executeUpdate(String sql) throws SQLException

PreparedStatement

- It is sub interface of Statement interface.
- To overcome limitations of statement we should use PreparedStatement object.
- Instantiation :

```
String sql = "";
PreparedStatement stmt = con.prepareStatement( sql );
```

- It compiles query only once and execute it multiple times.
- Using PreparedStatement object, we can not execute stored procedure and function.

CallableStatement

- It is sub interface of PreparedStatement interface.
- If we want to execute stored procedure or function then we should use Callable Statement object.
- Instantiation:

```
String sql = "....";
CallableStatement stmt = con.prepareCall(sql);
```

- Syntax to call Stored procedure

```
{call sp_procedure_name(arg1,arg2, ...)}
```

- Syntax to call Stored function

```
{?=call sf_function_name(arg1,arg2, ...)}
```

- If we want to execute stored procedure/function then we should use execute() method boolean execute() throws SQLException
- execute() method returnds boolean value. * true if the first result is a ResultSet object;
    - false if the first result is an update count or there is no result

MySQL Stored Procedure to insert book

```
DELIMITER $$
CREATE PROCEDURE sp_insert_book
(
pBookId INT,
pSubjectName VARCHAR(50),
pBookName VARCHAR(50),
pAuthorName VARCHAR(50),
pPrice FLOAT
)
BEGIN
    INSERT INTO BookTable VALUES
    ( pBookId, pSubjectName, pBookName,pAuthorName, pPrice );
END $$
DELIMITER ;
```

MySQL Stored Procedure to update book

```
DELIMITER $$
CREATE PROCEDURE sp_update_book
( pBookId INT, pPrice FLOAT )
BEGIN
    UPDATE BookTable SET price=pPrice WHERE book_id=pBookId;
END $$
DELIMITER ;
```

MySQL Stored Procedure to delete book

```
DELIMITER $$
CREATE PROCEDURE sp_delete_book
( pBookId INT )
BEGIN
   DELETE FROM BookTable WHERE book_id=pBookId;
END $$
DELIMITER ;
```

MySQL Stored Procedure to get all books

```
DELIMITER $$
CREATE PROCEDURE sp_select_book
(   )
BEGIN
  SELECT * FROM BookTable;
END $$
DELIMITER ;
```

IN and OUT parameter in Stored Procedure

```
CREATE TABLE accounts
(
    acc_number INT,
    name VARCHAR(50),
    balance FLOAT
);
```

```
INSERT INTO accounts VALUES(101, 'Rahul', 50000);

INSERT INTO accounts VALUES(102, 'Sandeep', 30000);
```

```
DELIMITER $$
CREATE PROCEDURE sp_fund_transfer
(
    IN srcAccountNumber INT,
    IN destAccountNumber INT,
    IN amount FLOAT,
    OUT srcBalance FLOAT,
    OUT destBalance FLOAT
)
BEGIN
    UPDATE accounts SET
    balance = balance - amount
    WHERE acc_number = srcAccountNumber;

    UPDATE accounts SET
    balance=balance + amount
    WHERE acc_number = destAccountNumber;

    SELECT balance INTO srcBalance
    FROM accounts
    WHERE acc_number = srcAccountNumber;

    SELECT balance INTO destBalance
    FROM accounts
    WHERE acc_number = destAccountNumber;
END $$
DELIMITER ;
```

JDBC Transaction

- If we perform any operation from java application then it is bu default auto commited.
- If we want to change that behavior then we should use setAutoCommit method.

```
con.setAutoCommit( false );
```

```
Connection connection = null;
Statement statement = null;
try
{
    connection = DBUtils.getConnection();
    statement =connection.createStatement();
    connection.setAutoCommit(false);

    String sql = "UPDATE accounts SET balance=balance - 5000 WHERE
acc_number=101";
    statement.executeUpdate(sql);

    sql = "UPDATE accounts SET balance=balance + 5000 WHERE
acc_number=102";
    statement.executeUpdate(sql);

    connection.commit();
    System.out.println("Fund transferd..");
}
catch( Exception ex )
{
    try
    {
        connection.rollback();
        ex.printStackTrace();
    }
```

```
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
```

ResultSet

- ResultSet object represents records returned by select query.
- Following are methods of java.sql.Connection interface:

1. Statement createStatement() throws SQLException
2. Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException
3. Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException

- ResultSet Fields

1. TYPE_FORWARD_ONLY
2. TYPE_SCROLL_INSENSITIVE
3. TYPE_SCROLL_SENSITIVE
4. CONCUR_READ_ONLY
5. CONCUR_UPDATABLE
6. HOLD_CURSORS_OVER_COMMIT
7. CLOSE_CURSORS_AT_COMMIT

- ResultSet Methods

1. boolean isBeforeFirst() throws SE
2. void beforeFirst() throws SE
3. boolean first() throws SE
4. boolean isAfterLast() throws SE
5. void afterLast() throws SE
6. boolean last() throws SE
7. boolean next() throws SE
8. boolean previous() throws SE
9. boolean absolute(int row) throws SE
10. boolean relative(int rows) throws SE
11. void updateRow() throws SE
12. void deleteRow() throws SE

13. void insertRow() throws SE

ResultSet Type

1. TYPE_FORWARD_ONLY
2. TYPE_SCROLL_INSENSITIVE
3. TYPE_SCROLL_SENSITIVE

ResultSet Concurrency

1. CONCUR_READ_ONLY
2. CONCUR_UPDATABLE

ResultSet Holdability

1. HOLD_CURSORS_OVER_COMMIT
2. CLOSE_CURSORS_AT_COMMIT

- By default, ResultSet object is forward only and read only.

```
Statement stmt = con.createStatement( );
```

- above statement is equivalent to

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_FORWARD_ONLY,
    ResultSet.CONCUR_READ_ONLY );
```

- How to make ResultSet Scollable?

```
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY );
```

- Or

```
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE );
```

- Following are methods of DatabaseMetaData interface which is used to check database support for following features:

1. boolean supportsResultSetType(int type)
2. boolean supportsResultSetConcurrency(int type, int concurrency) throws SQLException
3. boolean supportsResultSetHoldability(int holdability) throws SQLException

```
connection = DBUtils.getConnection();
DatabaseMetaData dm =
             connection.getMetaData();
```

Day 12

Multithreading

- Process of executing multiple task simultaneously is called multitasking.
- In context of oops, it called concurrency. It is minor pillar of oops.
- We can achive multitasking using process as well as thread.

> Why process based multitasking is called heavy weight multitasking?

- If CPU want to execute multiple processes simultaneously then it must first save state of current process into PCB then it can transfer control of itself to another process. It is called context switching.
- context switching is heavy task hence Process based multitasking is called heavy weight multitasking.

> Why thread based multitasking is called light weight multitasking?

- Thread always resides within process.
- To access resource assigned to the process, thread do not require context switching hence thread based multitasking is called light weight multitasking.

> Why java is multithreaded programming language?

- Thread is Non java / OS resource. To access OS thread, java application developer need not to use OS API. To access OS thread SUN/ORACLE developer has already developed ready framework. In other words java has given built in support to access OS thread hence every java application is multithread.
- When starts execution of java application, it also starts execution of main thread and garbage collector hence every java application is multithreaded.

Types of Thread

1. User Thread / Non Damemon Thread
    - User thread gets terminated immediatly after execution is over.
2. Daemon Thread / Background Thread
    - Daemon thread gets terminated when all its child thread gets terminated.

- If we create thread in java then by default it is considered as user thread. using setDaemon( ) method we can convert user thread into daemon thread.

```
Thread th = new Thread( ... );
th.setDaemon( true );
```

Thread Framework

- If we to use thread in java then we must import java.lang package.
- Interface
    1. Runnable
- Class(es)

    1. Thread
    2. ThreadLocal
    3. ThreadGroup
- Enum
    1. Thread.State
- Exception(s)
    1. InterruptedException
    2. IllegalMonitoStateException
    3. IllegalThreadStateException

Runnable

- If interface contains only one abstract method then it is called functional interface.
- Runnable is functional interface.
- "void run( )" is abstract method of Runnable interface.
- run() method is called business logic method.

Thread

- java.lang.Thread is managed version of unmanaged thread.
- In other words instance of java.lang.Thread is not a thread rather it presents operating system thread.
- JVM is resposible for mapping Java Thread instance to OS thread.

Nested Type of Thread

- Thread.State is enum declared inside Thread class.
- Enum constants of Thread state represents JVM thread states.
- Following are Thread States:
    1. NEW 0
    2. RUNNABLE 1
    3. BLOCKED 2
    4. WAITING 3
    5. TIMED_WAITING 4
    6. TERMINATED 5

```
State[] states = State.values();
for (State state : states)
{
    String name = state.name();
    int ordinal = state.ordinal();
    System.out.println(name+" "+ordinal);
}
```

Fields of Thread

1. public static final int MIN_PRIORITY = 1
2. public static final int NORM_PRIORITY = 5

3. public static final int MAX_PRIORITY = 10

- Thread fields represent thread priorites.

Constructors of Thread

1. public Thread()

```
Thread th1 = new Thread();
```

2. public Thread(String name)

```
Thread th1 = new Thread("UserThread#1");
```

3. public Thread(Runnable target)

```
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( target );
```

4. public Thread(Runnable target, String name)

```
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( target,"Abc" );
```

5.public Thread(ThreadGroup group, Runnable target)

```
ThreadGroup grp = new ThreadGroup("MyGrp");
Runnable target = new CThread(); //Upcasting
Thread th1 = new Thread( grp, target );
```

Methods of Thread Class

1. public static Thread currentThread()
2. public final String getName()
3. public final void setName(String name)
4. public final int getPriority()
5. public final void setPriority(int newPriority)
6. public Thread.State getState()
7. public final ThreadGroup getThreadGroup()
8. public void interrupt()
9. public final boolean isAlive()

10. public final boolean isDaemon()
11. public final void setDaemon(boolean on)
12. public final void join() throws InterruptedException
13. public static void sleep(long millis) throws InterruptedException
14. public void start()
15. public static void yield()

- If we want to use any hardware resource(CPU) efficiently then we should use thread.
- If we want to improve performance of application then we should void use of blocking calls in multithreaded application.
- Following calls are blocking calls:

1. sleep()
2. suspend()
3. join()
4. wait()
5. performing input operation

> What is the difference between sleep() and suspend( ) ?

- If we want to suspend execution of thread for specified milliseconds then we should use sleep() method.
- If we want to suspend execution of thread for infinite time then we should use suspend() method. In this case, by calling resume() method, suspended thred can come back to runnable state.
- Note : suspend() and resume() are deprecated.

```
Thread thread = Thread.currentThread();
System.out.println(thread.toString());
//Thread[main,5,main]
```

- thread.toString() returns a string representation of current thread, including
  - thread's name
  - priority
  - and thread group.

finalize() method

- It is non final method of java.lang.Object class
- Syntax: protected void finalize( ) throws Throwable
- If we want to release class scope resources then we should use finalize() method.
- If reference count of any instance is zero then it is eligible for garbage collection.
- Garbage Collector invoke finalize() method on instance whose reference count is zero.

Thread Creation

- In java, we can create thread using 2 ways

1. Using Runnable interface
2. Using Thread class

Thread Creation using Runnable

```
class CThread implements Runnable
{
        private Thread thread;
        public CThread( String name )
        {
                this.thread = new Thread( this );
                this.thread.setName( name );
                this.thread.start();
        }
        @Override
        public void run()
        {
                //TODO : Business Logic
        }
}
```

Thread Creation using Thread class

```
class CThread extends Thread
{
        public CThread( String name )
        {
                this.setName( name );
                this.start();
        }
        @Override
        public void run()
        {
                //TODO : Business Logic
        }
}
```

Realtion Between start() and run() method

- start() method do not call run() method
- If we call start() method on Thread instance (instance of java.lang.Thread class) then it is considered as request to JVM to register OS thread for Thread instance.
- When CPU schedular assign CPU to the OS thread then JVM invoke run() method on Runnable instance(argument pass to the Thread constuctor (this)).

Starting Thread

- If we want to start execution of thread then we should start() method.
- If we call start() method on thread instance multiple times then start() method throws IllegalThreadStateException.

```
Runnable target = new CThread();
Thread th = new Thread( target );
th.start();
```

Thread States

- If we create Thread instance then it is considered in NEW state.
- If we call start() method on Thread instance then it is considered in RUNNABLE state.
- If supend thread by calling Thread.sleep() then Thread is considered in TIMED_WAITING state.
- If control come out of run() method then Thread gets terminated.
- If thread gets terminated then it is considered in TERMINATED state.

> What is the difference between creating thread using Runnable and Thread?

- If class is sub class then we should create thread by implementing Runnable interface.
- If we create thread by implementing Runnable interface then every sub class must participate in threading behavior.
- If class is not a sub class then we should create thread by extending Thread class.
- If we create thread by extending Thread class then it is not mandatory for every sub class to participate in threading behavior. By overriding start() method, sub class can come out of threading behavior.

Thread Priority

- Managing thread is a job of CPU schedular.
- On the basis of Thread Priority, schedular assign CPU to the Thread.
- setPriority() method is used to set priority for Thread and getPriority() method is used to get priority of thread.

```
Thread thread = Thread.currentThread();
//thread.setPriority(thread.getPriority() + 3 ); //OK
thread.setPriority(Thread.NORM_PRIORITY + 3 ); //OK
System.out.println(thread.getName()+" : "+thread.getPriority());
```

- If the priority is not in the range MIN_PRIORITY to MAX_PRIORITY then setPriority method throws IllegalArgumentException
- Default priority of child thread depends on priority of parent Thread.
- If we set priority of a thread in java, then it gets mapped differently on Different operating system. Hence Same java application producess different behavior on different operating system
- Thread priorites makes java application platform dependant.

> Which feature of java make application platform dependant?

1. Thread Priorities
2. Abstract Window Toolkit(AWT) components

Thread Joining

- If we call join() method of thread instance then it blocks execution of all other threads.
- It is a blocking call.

```
Runnable target = new CThread();
Thread th = new Thread( target );
th.start();
th.join();
```

Resource Locking

- Without co-oridination, if multiple threads try to access shared resource then it is called as race condition.
- If we want to avoid race condition then we should lock the resource.
- synchronized is keyword in java which is used to lock the resource.
- If we use synchronized keyword with shared resource then it gets minitor object.
- Maintaining monitor object is expensive hence we should use it for minimum time otherwise waiting threads need to wait for long time which may degrade performance of application.
- Code written inside synchronized block or synchronized method is called critical section.
- If thread is waiting to get monitor object associated with shared resource then it is considered in Blocked state.

Inter Thread Communication

- Inter thread communication represents synchronization.
- With co-oridination, if multiple thread try to communicate with each other then it is called synchronization.
- Using same minitor object, if thread try to communicate with each other then it is called synchronization.
- If we want to achive synchronization then we should use wait, notify/notifyAll() methods.
- It is mandatory to call above methods from synchronized block / synchronized method.
- If we try to call these methods from unsynchronized block/method then these methods throws IllegalMonitoStateException.

Day 13

Reflection

- Data about data / data which describes another data is called metadata.
- Applications of metadata

1. After compilation, java compiler generates bytecode and metadata. Due to this metadata, there is no need to include header file in .java file.
2. To display class information in intelisense window IDE implicitly use metadata.
3. Metadata help JVM tp serialize or clone state of java instance
4. To keep track of lifetime of object, garbage collector implicitly use metadata.

- If we want to analyze and process metadata then we should use reflection.

- Reflection is java language feature which provides type(s) that is used to analyze and process metadata.

- If we want to use Reflection then we should use types declared in java.lang and java.lang.reflect package.

- Types declared in java.lang:

1. Class

- Types declared in java.lang.reflect:

1. Array
2. Constructor
3. Field
4. Method
5. Modifier
6. Parameter

- Application of Reflection

1. javap is java language disassembler. It is used to show type information using F.Q. type name. It implicitly use reflection
2. To read metadata from .class file IDE implicitly use Reflection.
3. To access values of private field debugger implicitly use reflection.
4. For object mapping, hibernate implicitly use reflection.
5. To implement drag and drop feature, IDE implicitly use reflection

```
class Date{ }
class Address{  }
class Person
{
    String name;
    Date birthDate;
    Address currentAddres;
}
```

```
class Program
{
    public static void main(String[] args)
    {
        Person p = new Person();
    }
}
javac Program.java
java Program
```

- Class is a final class declared in java.lang package.
- Instances of the class java.lang.Class represent classes and interfaces in a running Java application.
- java.lang.Class has no public constructor. Instead Class objects are constructed automatically by the Java Virtual Machine.

How to get reference of Class class instance?

1. Using getClass() method.

```
Integer n1 = new Integer(125);
Class<?> c = n1.getClass();
```

2. Using .class syntax

```
Class<?> c = Number.class;
```

3. Using Class.forName

```
System.out.print("F.Q.Class Name");
String className = sc.nextLine();
//java.lang.Thread
Class<?> c = Class.forName(className);
```

File IO

- File is container that is used to store record permanantly on HDD.
- Stream is an abstraction(object) that is used to produce(write) and consume(read) information from source to destination.
- If we want to do file handling we should use types declared in java.io package.
- Interfaces

1. Closeable
2. Flushable
3. FilenameFilter

4. DataInput
5. DataOutput
6. ObjectInput
7. ObjectOutput
8. Serializable

- Classes

1. Console
2. File
3. InputStream
4. OutputStream
5. FileInputStream
6. FileOutputStream
7. BufferedInputStream
8. BufferedOutputStream
9. DataInputStream
10. DataOutputStream
11. ObjectInputStream
12. ObjectOutputStream
13. PrintStream
14. Reader
15. Writer
16. FileReader
17. FileWriter
18. BufferedReader
19. BufferedWriter
20. InputStreamReader
21. OutputStreamWriter
22. PrintWriter

Binary File

- e.g .class, .obj, .mp3, .jpg etc.
- If we read binary file then we must use specific program.
- Binary require less processing hence it is faster than text file.
- It doesnt save data in human readable format.
- In java, InputStream, OutputStream and their sub classes are used to manipulate binary file.

Text File

- e.g .java, .txt, .rtf, .doc, .xml etc.
- We can read text file using any text editor.
- Text file require more processing hence it is slower than binary file
- It can save data in human readable format
- Reader, Writer and their sub classes are used to manipulate text file.

java.io.File

- It is a java class whose instance represent operating System file, directory or drive.
- Use:
    1. To create empty file / empty directory
    2. To read metadata of OS File, Directory and Drive.

Socket Programming

http://www.sunbeaminfo.com:8080/Dac/Index.html Protocol : http
Host Name : www.sunbeaminfo.com Port Number : 8080 Path Name : /Dac/Index.html